Enforcing Safety and Consistency Constraints in Policy-Based Authorization Systems

ADAM J. LEE
University of Pittsburgh
and
MARIANNE WINSLETT
University of Illinois at Urbana-Champaign

In trust negotiation and other forms of distributed proving, networked entities cooperate to form proofs of authorization that are justified by collections of certified attribute credentials. These attributes may be obtained through interactions with any number of external entities and are collected and validated over an extended period of time. Although these collections of credentials in some ways resemble partial system snapshots, current trust negotiation and distributed proving systems lack the notion of a consistent global state in which the satisfaction of authorization policies should be checked. In this article, we argue that unlike the notions of consistency studied in other areas of distributed computing, the level of consistency required during policy evaluation is predicated solely upon the security requirements of the policy evaluator. As such, there is little incentive for entities to participate in complicated consistency preservation schemes like those used in distributed computing, distributed databases, and distributed shared memory. We go on to show that the most intuitive notion of consistency fails to provide basic safety guarantees under certain circumstances and then propose several more refined notions of consistency that provide stronger safety guarantees. We provide algorithms that allow each of these refined notions of consistency to be attained in practice with minimal overheads and formally prove several security and privacy properties of these algorithms. Lastly, we explore the notion of strategic design trade-offs in the consistency enforcement algorithm space and propose several modifica-

This research was supported by the NSF under grants IIS-0331707, CNS-0325951, and CNS-0524695 and by Sandia National Laboratories under grant number DOE SNL 541065. A.J. Lee was also supported in part by a Motorola Center for Communications graduate fellowship.
A preliminary version of this article appears in the *Proceedings of the 13th ACM Conference on Computer and Communications Security* [Lee and Winslett 2006].
This work was carried out while A.J. Lee was at the University of Illinois at Urbana-Champaign.
Authors' addresses: A. J. Lee, Department of Computer Science, University of Pittsburgh, 210 S. Bouquet St., Pittsburgh, PA 15260; email: adamlee@cs.pitt.edu; M. Winslett, Department of Computer Science, University of Illinois at Urbana-Champaign, 201 N. Goodwin Ave., Urbana, IL 61801; email: winslett@cs.uiuc.edu.

tions to the core algorithms presented in this article. These modifications enhance the privacy-preservation or completeness properties of these algorithms without altering the consistency constraints that they enforce.

Categories and Subject Descriptors: C.2.4 [**Distributed Systems**]: Distributed applications; D.4.6 [**Operating Systems**]: Security and Protection—*access controls, authentication*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security

Additional Key Words and Phrases: consistency, credentials, distributed proving, trust negotiation

ACM Reference Format:
Lee, A. J. and Winslett, M. 2008. Enforcing safety and consistency constraints in policy-based authorization systems. ACM Trans. Inf. Syst. Secur. 12, 2, Article 8 (December 2008), 33 pages. DOI = 10.1145/1455518.1455520. http://doi.acm.org/10.1145/1455518.1455520.

1. INTRODUCTION

It is difficult to design flexible and secure authorization systems for environments in which trust relationships cannot be determined a priori. Two proposed authorization techniques for these types of environments are trust negotiation [Becker and Sewell 2004; Bertino et al. 2004; Koshutanski and Massacci 2005; Li et al. 2005; Li and Michelle 2003; Winsborough and Li 2002; Winslett et al. 2002; and Yu et al. 2003] and distributed proving [Bauer et al. 2005; Minami and Kotz 2006; and Winslett et al. 2005]. In these types of systems, participants collect certified credentials that describe their attributes, environmental conditions, and other state information from any number of external entities. These credentials can then be used when attempting to satisfy the authorization policies protecting sensitive resources in the system.

To some extent, the collection of credentials used to satisfy a given authorization policy acts as a partial snapshot of the system within which the policy is evaluated. This is an abuse of terminology, however, as this "snapshot" is collected over a variable-length window of time and thus may not actually represent a system state that ever existed; to avoid confusion, in this article we will refer to these collections of credentials as *views*. Clearly, the correctness of an authorization decision depends on the validity and stability of the view used during policy evaluation. If we assume that each credential is stable (i.e., that the assertion stated in the credential remains true until its preordained expiration time) then policy evaluation can be reduced to the problem of stable predicate evaluation on distributed snapshots [Chandy and Lamport 1985]. However, because it is possible for credentials to become invalidated prematurely, this somewhat naive model of policy evaluation can erode the safety guarantees of the underlying authorization system. That is, the satisfaction of a policy in a naive decentralized authorization model does not necessarily imply that all credentials used as evidence to satisfy the policy were ever simultaneously valid, let alone simultaneously valid at the time when the policy was determined to be satisfied. This is in stark contrast to centralized authorization systems in which a more transactional semantics for policy

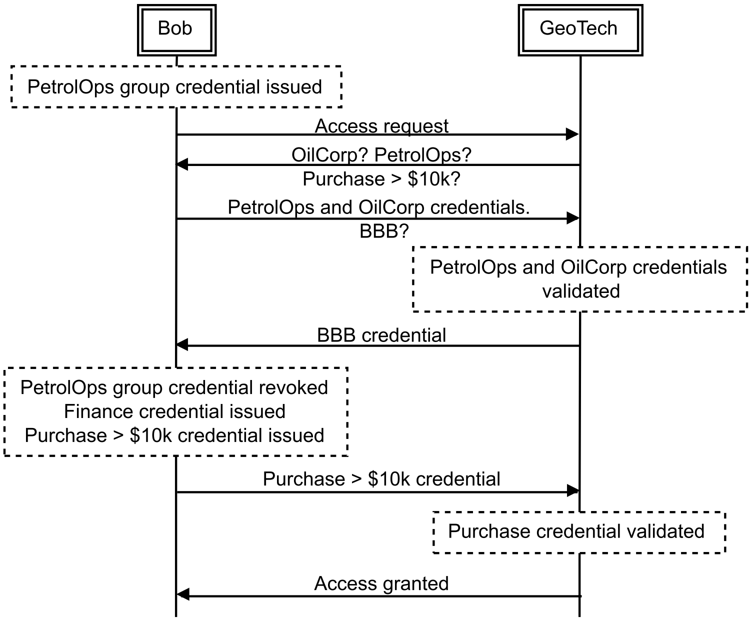
Bob
GeoTech
PetrolOps group credential issued
Access request
OilCorp? PetrolOps?
Purchase > $10k?
PetrolOps and OilCorp credentials.
BBB?
PetrolOps and OilCorp credentials validated
BBB credential
PetrolOps group credential revoked
Finance credential issued
Purchase > $10k credential issued
Purchase > $10k credential
Purchase credential validated
Access granted

Fig. 1. A graphical representation of Bob's interaction with GeoTech.

evaluation can be easily enforced. This relaxation of the semantics of policy satisfaction is especially worrisome in trust negotiation and distributed proving protocols, as interactions in these types of systems typically involve multiple rounds of interaction and credential exchange. Consider the following two examples of the problems that can be caused by unstable credentials.

Example 1. Figure 1 illustrates one case in which inconsistent credential state can cause undesirable decisions to be made. In this scenario, Bob works in the finance department of Acme Petroleum Corporation (APeC), though he also spends part of his time on loan to the Petroleum Operations group helping manage their operational budget. While consulting for the operations group, Bob is given a PetrolOps group credential to allow him basic access to the operations group's resources. To speed up some of his research, Bob wishes to access an online geological database provided by GeoTech, a third-party vendor. GeoTech allows operations group members at Department of Energy certified oil companies who are authorized to make purchases of over $10,000—which is the cost of a department subscription to the database—trial access to their service. Bob submits his PetrolOps group credential and APeC's OilCorp credential to GeoTech along with a policy stating that it must provide proof of membership in the Better Business Bureau to see his purchase authorization. GeoTech verifies Bob's PetrolOps credential and APeC's OilCorp credential and then sends Bob its BBB credential. As a consultant to the operations group, Bob is not authorized to make purchases of more than $200, so he should not be able to satisfy this policy. However, Bob can make purchases of this size for

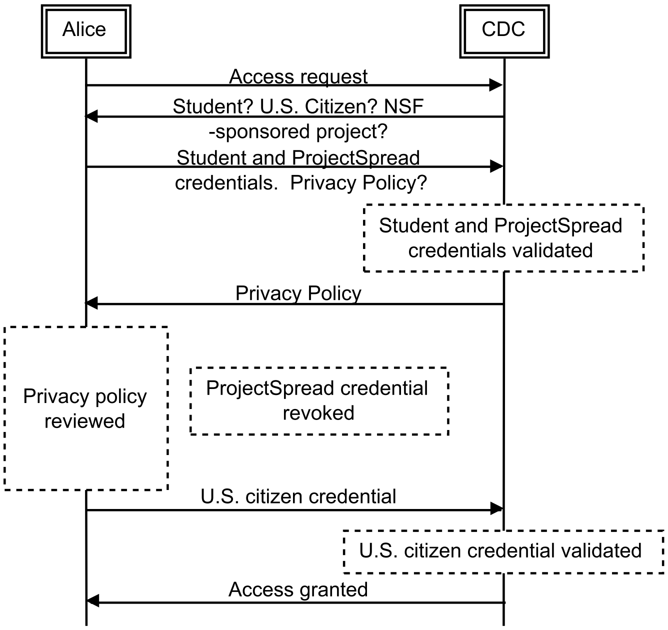
Alice
CDC
Access request
Student? U.S. Citizen? NSF -sponsored project?
Student and ProjectSpread credentials. Privacy Policy?
Student and ProjectSpread credentials validated
Privacy Policy
Privacy policy reviewed
ProjectSpread credential revoked
U.S. citizen credential
U.S. citizen credential validated
Access granted

Fig. 2. A graphical representation of Alice's interaction with the CDC.

the finance group. Bob then activates his Finance group credential (which invalidates his PetrolOps credential) and obtains a certified Purchase attestation authorizing him to make purchases of up to $10,000 dollars, which he then submits to GeoTech. GeoTech verifies this credential and grants Bob access to the database. The inconsistent system view used by the database allows Bob to convince GeoTech that he is an operations group member who is allowed to make purchases of over $10,000 when he is actually either an operations group member or a finance group member who is allowed to make purchases of over $10,000.

Example 2. Figure 2 shows how premature credential revocations can lead to inconsistencies that alter the expected semantics of policy satisfaction. Alice is a Ph.D. student studying infectious diseases at State University. As part of her research, Alice wishes to access an outbreak incident database hosted by the Center for Disease Control. The CDC requires that academic users of this data be U.S. citizens and members of an NSF-sponsored epidemiology project. To this end, Alice discloses her Student credential issued by State University and her ProjectSpread credential issued by the NSF. Alice considers her citizenship private, however, and requires that she first receive a certified privacy policy that she manually reviews prior to releasing her citizenship credential. Alice submits a policy to this effect to the CDC. The CDC verifies Alice's Student and ProjectSpread credentials and then discloses its certified PrivacyPolicy to Alice. Just then, Alice's research adviser calls and notifies her that effective immediately, she will no longer be supported by the Spread project; the NSF then revokes her ProjectSpread credential. Alice then reviews the PrivacyPolicy submitted by the CDC and decides that it is safe to disclose her USCitizen credential. The CDC verifies this credential and permits Alice to access the requested data, as it did not detect that her project membership had been revoked prior to policy satisfaction.

The safety problems that emerged in the above examples occur because credentials are collected over a noninstantaneous window of time. In general, credential and policy instabilities can arise from one or more of the following four causes. First, the *natural expiration* of a credential can cause problems if a previously-valid credential expires before other required credentials can be validated. Second, *inter-credential dependencies* can give rise to problems if, for example, the activation of a new role causes the revocation of a previously activated role (as in Example 1). Third, an *external event* might cause the invalidation of a certain credential after it is validated, but prior to the entire policy being satisfied. For example, the removal of Alice from the Spread project in Example 2 caused credential revocation. Lastly, an *unstable environment* could cause policy instability if the policy is predicated on some aspect of the environment, such as the time of day or occupancy status of a room.

To the best of our knowledge, the problem of enforcing view consistency in trust negotiation and distributed proving systems has not been discussed elsewhere in the literature. Though similar to the consistency problems studied in distributed systems [Tanenbaum and van Steen 2002], distributed databases [Cellary et al. 1988], and distributed shared memory [Adve and Gharachorloo 1996], this problem is in many ways their dual. In these previous works, ensuring a consistent global state has been the concern of both data providers and users, as many entities can update the values of data fields replicated at a number of sites; this provides all parties with the incentive to cooperate. However, since a credential revocation can be made only by the issuer of that credential (and thus consistent update sequences can be attained trivially), the problem studied in this article becomes the concern only of data consumers. In fact, the degree to which each data consumer is concerned with this problem may even vary based on the criticality of the policy being evaluated. For instance, a hardware store offering a discount to students of a particular university will probably not be concerned if a student ID credential is revoked after it has been issued for the semester, much less if it is revoked during a policy evaluation; an electronic door lock protecting access to expensive laboratory equipment at the university would care, however. Heavy-weight solutions that require the cooperation of groups of certificate authorities (CAs) and users are not suitable, as the consistency property required will vary from user to user and preserving the autonomy of entities in the open system is of the utmost importance.

In this article, we make several contributions regarding the level of safety attainable when evaluating policies in authorization systems that employ trust negotiation or other forms of distributed proving. To the best of our knowledge, we present the first formalization of the view consistency problem for trust negotiation and distributed proving systems and show how naive approaches to policy evaluation can lead to the permission of undesirable accesses to system resources in the face of prematurely invalidated credentials (Section 2). We then define several levels of credential view consistency, each of which provides different guarantees on the types of inappropriate access conditions that can be prevented (Section 3). We provide algorithms that can be incorporated into existing trust negotiation and distributed proving systems to attain these levels of consistency and prove the correctness of each algorithm (Section 4). We also demonstrate other desirable characteristics of these algorithms and their extensions, including the fact that they require only minimal cooperation between the users engaged in the the trust negotiation or distributed proving protocol and no cooperation between groups of CAs or other users (Section 5). Finally, we comment on previous related work (Section 6) and examine potential areas for future work (Section 7).

2. SYSTEM ASSUMPTIONS AND PROBLEM DEFINITION

In this section, we present our assumptions regarding the open systems in which trust negotiation and distributed proving protocols are used. We then formally describe the problem of determining the consistency level of a system view used to evaluate an authorization policy.

2.1 System Model

An open system consists of a possibly infinite set $\mathcal{E}$ of entities, each of which is a resource provider, client, or both. *Resource providers* are entities who wish to offer resources or services to other entities in the system, while *clients* are entities that access the functionality offered by resource providers. Resource providers may wish to enforce authorization checks on the resources or services that they provide; trust negotiation or distributed proving will be used for this purpose, as the lack of preexisting trust relationships in the system prevents the use of traditional identity-based authorization mechanisms. Note that a given resource provider may also be a client of other resource providers.

We place no limitations on the temporal duration of a trust negotiation or distributed proving session other than those imposed by the underlying protocol. For example, many trust negotiation protocols halt if no measurable progress is made during a particular round of the negotiation [Li et al. 2005; Yu et al. 2003]; we do not prevent this, nor do we require any such constraints be in place. Unless explicitly stated to the contrary, we assume that the credentials used by an entity during the execution of one of these protocols may be obtained dynamically at runtime. This assumption allows portions of a distributed proof to be “outsourced” to other entities (as in Bauer et al. [2005]; Minami and Kotz [2006]; and Winslett et al. [2005]) and permits entities to acquire new attribute certificates while a trust negotiation session is in progress. These assumptions indicate that the collection of credentials used as the view in which an authorization policy is satisfied may be composed of the observations of an arbitrary number of entities and be collected over a variable-width window of time.

We assume that the certified attribute and environmental state information used to satisfy trust negotiation policies or form distributed proofs will be issued by an arbitrary number of CAs that exist in the system. All credentials issued will have an expiration time but may also be revoked prematurely by the issuing CA (as was the case with Alice's ProjectSpread credential in Section 1). In the remainder of this article, we will denote the set of all credentials by $\mathcal{C}$ and the set of all policies by $\mathcal{P}$. Given a credential $c \in \mathcal{C}$, we denote by $\alpha(c)$ the earliest time at which the issuing CA would possibly consider c to be valid. In the case of X.509 certificates [Housely et al. 1999], $\alpha(c)$ would be the time indicated in the "Not Before" field of the certificate; if no such field exists, then $\alpha(c)$ indicates the issue time of the credential. Similarly, we denote the expiration time of a credential c by $\omega(c)$.

We assume that once a credential is revoked, it will never again become valid. For example, if Bob wishes to again activate his membership in the operations group role after it is revoked (see Example 1), his previously-revoked PetrolOps credential cannot again be used, he must obtain a new PetrolOps credential. Since only the issuing CA may revoke a credential, each CA can ensure that an omniscient view of the credentials that it has issued remains consistent at all times. We assume that each CA offers an online method that allows any entity to check the current status of a particular credential issued by the CA. This functionality could be provided through the Online Certificate Status Protocol (OCSP) [Myers et al. 1999] or by an online CA such as COCA [Zhou et al. 2002].

2.2 Problem Definition

Prior to accepting a given credential as evidence that can be used to satisfy some portion of an authorization policy, the policy evaluator must first verify that the credential is valid. In this article, we are concerned with two types of credential validity: syntactic and semantic.

Definition 2.1 *Syntactic Validity*. A credential c is syntactically valid if the following conditions hold: (i) it is formatted properly, (ii) it has a valid digital signature, (iii) the time $\alpha(c)$ has passed, and (iv) the time $\omega(c)$ has not yet passed.

Definition 2.2 *Semantic Validity*. A credential c is semantically valid at time t if an online method of verifying c's revocation status (e.g., by using OCSP or COCA) indicates that c was not revoked at some later time t' such that $\alpha(c) \leq t \leq t'$.

Informally, if a credential is syntactically valid, it is well-formed. The semantic validity of a credential at a given time implies that the credential has not been revoked by its issuer prior to that time; that is, the credential issuer asserts that the meaning of the credential is still valid. To ground these definitions with a real-world example, in the case of credit card validation, verifying syntactic validity involves checking that the signature on the back of the card matches the signature on the charge slip, the card has an appropriate issuer logo on the front, and the expiration date has not passed. Semantic validation occurs when the credit card clearinghouse authorizes a transaction. Note that in the case that a credential is assumed to be a stable assertion, syntactic validity implies semantic validity. We now define the more general concept of validity and derive two propositions and a corollary that will be useful later in the article.

Definition 2.3 *Validity*. A credential c is valid at time t if it is both syntactically and semantically valid at time t.

PROPOSITION 2.4. *If a credential c is found to be syntactically valid at a time t' such that $\alpha(c) \leq t' < \omega(c)$, then c is syntactically valid at all times t where $\alpha(c) \leq t < \omega(c)$.*

PROPOSITION 2.5. *If a credential c is semantically valid at a time $t' \geq \alpha(c)$, then c is semantically valid at all times t where $\alpha(c) \leq t \leq t'$.*

COROLLARY 2.6. *If a credential c is valid at a time t' such that $\alpha(c) \leq t' < \omega(c)$, then c is valid at all times t where $\alpha(t) \leq t \leq t'$.*

As was observed earlier, each credential collected by an entity during a trust negotiation or distributed proving protocol constitutes a piece of evidence attesting to a small portion of the global state of the network. During a trust negotiation or the construction of a distributed proof, these pieces of evidence are collected over time and used to incrementally satisfy a given authorization policy. We now more precisely define one entity's view of the system in terms of the credentials acquired during a particular trust negotiation or distributed proving session.

Definition 2.7 *Credential State*. Let the set T contain all possible timestamps and the null value $\perp$. The state of a credential c as observed by an entity e is defined as $s = \langle c, r, syn, sem_v, sem_i\rangle \in \mathcal{C} \times (T \setminus \{\perp\}) \times \mathrm{B} \times T \times T$. The value r indicates the local time at which c was received by e. The Boolean value syn is true if c is syntactically valid, false otherwise. The values sem_v and sem_i denote the most recent time that c was verified to be semantically valid and the *first* time that c was found to be semantically invalid, respectively. If the semantic validity of c has not yet been checked, both sem_v and sem_i will be set to $\perp$, otherwise at least one of these fields will contain a non-null timestamp from the set $T \setminus \{\perp\}$. We use $\mathcal{S}$ to denote the set of all possible credential state tuples. Throughout this article, we will use dot notation to access fields of these state tuples (e.g., $s.r$ represents the receipt time of the credential whose state is stored in s).

Definition 2.8 *View*. A set of credential states observed by an entity e is called one of e's views of the system. A view contains at most one credential state tuple for any particular credential c.

Given the above definitions, we now have a precise vocabulary for describing an entity's knowledge about the state of the system. Since this state information is gathered over time, it cannot be considered to be a precise snapshot

Table I. A Summary of Notation Introduced in Section 2

—$\mathcal{E}$ is the set of all entities —$\mathcal{C}$ is the set of all credentials —$\mathcal{P}$ is the set of all policies —T is the set of all timestamps —$\alpha(c)$ denotes the start of c's validity period —$\omega(c)$ denotes c's expiration time —$V_e^{P,t}$ is a view containing state information for credentials that entity e considers relevant to policy P at time t	—$s \in \mathcal{S}$ is a credential state tuple in which: —$s.c$ is a credential —$s.r$ is the time at which c was received —$s.syn$ indicates c's syntactic validity status —$s.sem_v$ is the most recent time at which c was observed to be semantically valid —$s.sem_i$ is the first time at which c was observed to be semantically invalid

of the global state and thus the consistency of an entity's view of the system becomes important to consider.

Definition 2.9 *Relevance*. A credential c is considered relevant to a policy P by entity e at time t if e has received c and e's negotiation strategy considers the satisfaction of P in some way dependent on c at time t. Given a view V_e observed by entity e, $V_e^{P,t}$ is the subset of V_e containing state information for credentials that e considers to be relevant to P at time t.

Definition 2.10 *View Consistency*. A view $V_e^{P,t}$ is ϕ-consistent if $V_e^{P,t}$ satisfies a predicate ϕ that places temporal constraints on the times at which e observes the validity of each credential c whose state information is stored in $V_e^{P,t}$.

Definition 2.9 is very subtle, as the concept of relevance will be different for different negotiation strategies. Given the history of a trust negotiation session, a trust negotiation strategy examines an incoming message containing some number of credentials and policies to then determine the next step to be taken during the negotiation process [Yu et al. 2003]. Thus, every trust negotiation strategy implicitly defines its own concept of relevance. Further, the set of credentials considered relevant to a policy P by a particular strategy might change over time as multiple ways of satisfying a given policy are attempted. For example, when evaluating the policy $P = c_1 \wedge (c_2 \vee c_3)$, a strategy may initially consider c_1 and c_2 relevant to P and determine whether a consistent view can be constructed using these credentials. If this fails, then it may backtrack and decide that c_1 and c_3 are relevant to P and again attempt to construct a consistent view.

Definition 2.10 fundamentally ties together the concepts consistency and relevance. As a result, the notion of consistency can thus be undermined by a faulty interpretation of relevance (for instance, by assuming that nothing is relevant to P). At a minimum, a strategy should consider the set of credentials used to satisfy P to be relevant to P and may also include other credentials stored in its local state in this set (for instance, credentials used to satisfy the release policies protecting credentials disclosed during the authorization protocol invoked to satisfy P). Only the negotiation strategy has the autonomy and local knowledge necessary to decide which credentials are relevant at each moment and should thus be subjected to consistency requirements.

As a convenience to the reader, Table I contains a summary of the notation introduced in this section.

2.3 Practical Considerations for Consistency Enforcement

In this article, we focus on limiting the unexpected behaviors that trust negotiation and distributed proving systems can manifest as a result of inconsistent views. To this end, we define several enforceable notions of view consistency, discuss the guarantees provided by each, and provide algorithms to attain these levels of view consistency in practice. In proposing practical mechanisms for view consistency enforcement, we will keep several high-level requirements in mind.

2.3.1 *Loose clock synchronization.* A minimal level of clock synchronization is necessary, as otherwise the expiration times stored in credentials could not be reliably interpreted. However, we cannot assume that clocks are closely synchronized (e.g., at second-level precision).

2.3.2 *Minimal cooperation.* View consistency is a concern only for the policy evaluator. We cannot assume that groups of CAs, groups of CAs and users, or large groups of users will be willing to cooperate, as there is no incentive for this.

2.3.3 *Minimal impact to existing protocols.* Trust negotiation and distributed proving have been active areas of research over the course of the last several years. To ensure that the work done in these areas remains usable, view consistency enforcement should require minimal changes to existing trust negotiation and distributed proving protocols.

We will bear these requirements in mind throughout the remainder of this article; in Section 5 we will discuss the ways in which our solutions for enforcing view consistency satisfy these requirements.

3. LEVELS OF CONSISTENCY

In this section, we present four increasingly more powerful levels of view consistency. We show that the guarantees afforded by each of these consistency levels can be strengthened if assumptions can (safely) be made about which of the four reasons for credential invalidation described in Section 1 can be expected to apply during the course of the authorization protocol. This indicates that like many other aspects of trust negotiation and distributed proving, the choice of consistency level required is likely to be a strategic choice made independently by each protocol participant. We defer all discussion pertaining to unstable environments until Section 5.

3.1 Incremental Consistency

Most people have an intuitive understanding of how to satisfy a policy: present evidence that each clause of the policy is satisfied. For instance, if Alice wishes to cash a check and is asked for two forms of ID, she could, for example, produce a driver's license and a passport during her transaction with the bank teller. The teller can verify that both IDs show Alice's picture and list the same home address and thus be reasonably satisfied that Alice is indeed who she says that she is. The teller is convinced that her view of the system is consistent because Alice could produce valid instances of the required documents during the course of their interaction. We call this intuitive notion of consistency *incremental consistency*. To formally define incremental consistency, we first define the predicates $\mathit{checked} : \mathcal{S} \rightarrow \mathrm{B}$ and $\phi_{inc} : 2^{\mathcal{S}} \rightarrow \mathrm{B}$.

$$\mathit{checked}(s) \equiv (s.syn = true) \wedge (s.sem_v \neq \perp) \quad (1)$$

$$\phi_{inc}(V) \equiv \forall s \in V : \mathit{checked}(s) \wedge (\alpha(s.c) \leq s.r \leq s.sem_v) \quad (2)$$

The predicate $\mathit{checked}(s)$ is satisfied if and only if the syntactic validity of $s.c$ has been verified and $s.c$ was ever observed to be semantically valid. The predicate $\phi_{inc}(V_e)$ is satisfied if and only if each credential in the view V_e was valid at the point that it was received by e. Note that Corollary 2.6 is used when computing the endpoints of each credential's observed validity period. Thus, the formal definition of incremental consistency is as follows.

Definition 3.1 *Incremental Consistency*. A view $V_e^{P,t}$ is incrementally consistent if and only if $\phi_{inc}(V_e^{P,t})$ is true.

Incremental consistency works for Alice and the bank teller, as it is exceedingly unlikely that Alice's driver's license or passport will be revoked or become invalid during their transaction. In addition to being intuitively useful, incremental consistency is also widely used in practice. Current trust negotiation prototypes (e.g., Becker and Sewell [2004], Bertino et al. [2004], Koshutanski and Massacci [2005], and Winslett et al. [2002]) implement incremental consistency by validating credentials as they are received. This approach to credential validation is also discussed in many papers that present protocols and strategies for trust negotiations and distributed proving that, to the best of our knowledge, have not yet been implemented (e.g., Bonatti and Samarati [2000], Li et al. [2005], Winsborough and Li [2002], and Winslett et al. [2005] to name a few).

Incremental consistency works especially well when authorization policies are stable predicates, such as "Alice has paid her 2005 income taxes" or "process X has terminated." If all relevant user attributes and environmental conditions are stable, then incremental consistency allows us to conclude that all credentials used to satisfy a given policy were simultaneously valid at the time of policy satisfaction. This, of course, assumes that we verify that no credential expired naturally before the final decision was made.

If policy predicates are not stable, however, incremental consistency cannot guarantee that all relevant credentials were ever valid simultaneously. For example, recall Example 1 presented in Section 1. Figure 3 shows GeoTech's view of Bob's credentials in this system, where the validity periods of each credential are indicated with horizontal lines. GeoTech never observed Bob's PetrolOps and Purchase credentials to be valid simultaneously. With intercredential dependencies, such as that between Bob's PetrolOps and Finance credentials, incremental consistency is not always a good choice.

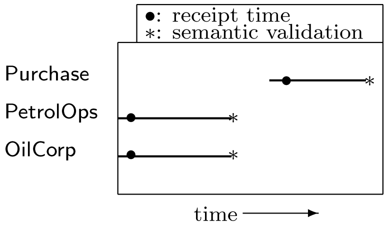

Fig. 3. An incrementally consistent view.

Although incremental consistency is the only form of view consistency supported by existing trust negotiation prototypes, we believe that this is only because until now the issue of view consistency has not received any attention. The trust negotiation and distributed proving literature is full of examples motivating the use of these systems in computing grids, dynamic coalitions, and ubiquitous computing environments. These environments are all highly dynamic and, in some cases, could involve the use of mutually-exclusive roles and access rights; under these conditions incremental consistency is likely to be unsatisfactory. We now present three stronger notions of view consistency that are easily enforceable in practice and discuss the guarantees that each provides.

3.2 Internal Consistency

In this section, we define and discuss a stronger notion of view consistency that we will call *internal consistency*. Informally, if an authorization decision is made using an internally consistent view, then all credentials relevant to the authorization decision were valid *simultaneously* at some point in time during the authorization protocol. To formally define internal consistency, we first define the functions $start : 2^{S} \rightarrow T$ and $end : 2^{S} \rightarrow T$, and the predicate $\phi_{int} : 2^{S} \rightarrow \mathrm{B}$.

$$start(V) = min(\{s.r \mid s \in V\}) \tag{3}$$

$$end(V) = max(\{s.r \mid s \in V\}) \tag{4}$$

$$\begin{aligned} \phi_{int}(V) \equiv\ & (\forall s \in V : checked(s)) \\ & \wedge (max(\{\alpha(s) \mid s \in V\}) < min(\{s.sem_i \mid s \in V\})) \\ & \wedge (max(\{\alpha(s) \mid s \in V\}) < end(V)) \\ & \wedge (min(\{\omega(s) \mid s \in V\}) > start(V)) \end{aligned} \tag{5}$$

The function $start(V)$ is the earliest local time at which a credential in V was received; similarly, $end(V)$ is the latest local time at which a credential in V was received. For a given view, V, these functions effectively bound the duration of the interactive portion of the associated authorization protocol. The predicate ϕ_{int} holds true if and only if (i) each credential in the view was at one point observed to be valid, (ii) the last credential to become valid does so before

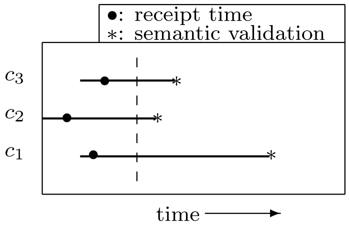

Fig. 4. An internally consistent view.

the minimum known endpoint of any credential's validity period, (iii) the last credential to become valid does so before the end of the authorization protocol, and (iv) the minimum known endpoint of any credential's validity period occurs after the start of the authorization protocol.

Definition 3.2 *Internal Consistency*. A view $V_e^{P,t}$ is internally consistent if and only if $\phi_{int}(V_e^{P,t})$ is true.

Internal consistency does not imply that all relevant credentials used to satisfy a policy are valid simultaneously at the moment the policy is decided to be satisfied. Rather, it implies that all relevant credentials are valid simultaneously at some point during the authorization protocol. Given a graphic representation of an internally consistent view, one should be able to draw at least one vertical line that intersects each credential's validity interval (see Figure 4).

Internal consistency is an important consistency level to consider for several reasons. Most importantly, by ensuring that all credentials used during a protocol were simultaneously valid, it prevents attacks in which one party evades mutual exclusion constraints by strategically interleaving role activation and deactivation events with credential disclosures (see Example 1). Further, internal consistency offers flexibility, as it does not constrain the point in time during the protocol at which all credentials must be simultaneously valid; this offers client entities the ability to deactivate roles as required by events external to the authorization protocol without compromising the consistency of the view used by their protocol execution partner. Lastly, if external events cannot cause the revocation of a credential, then all credentials in an internally consistent view can be shown to be valid at the time of policy satisfaction. However, should an external revocation occur, this is not the case. Recall Example 2, in which all of Alice's credentials were valid at the start of the authorization protocol, but due to the NSF's revocation of her ProjectSpread credential, they were not all valid at the time that the decision was made.

3.3 Stronger Levels of Consistency

In some cases, it might be desirable not only to have the guarantee that each relevant credential in a given view was valid simultaneously at some point during the authorization protocol, but also that they were all valid

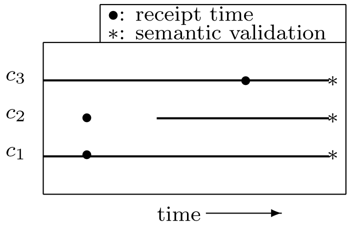

Fig. 5. An endpoint consistent view.

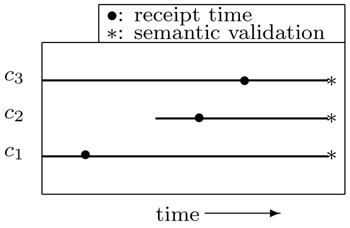

Fig. 6. An interval consistent view.

simultaneously at the endpoint of the authorization protocol. In others, perhaps it is required that each relevant credential is valid from the time that it is received until the decision point of the authorization protocol, as this may imply some level of stability in the system. We will call these levels of consistency *endpoint consistency* and *interval consistency*, respectively (see Figures 5 and 6). These consistency levels are defined in terms of the $\phi_{end} : 2^{\mathcal{S}} \to \mathrm{B}$ and $\phi_{interval} : 2^{\mathcal{S}} \to \mathrm{B}$ predicates.

$$\phi_{end}(V) \equiv \forall s \in V : checked(s) \wedge (\alpha(s.c) \leq end(V) \leq s.sem_v) \qquad (6)$$

$$\phi_{interval}(V) \equiv \forall s \in V : checked(s) \wedge (\alpha(s.c) \leq s.r \leq end(V) \leq s.sem_v) \qquad (7)$$

Definition 3.3 *Endpoint Consistency*. A view $V_e^{P,t}$ is endpoint consistent if and only if $\phi_{end}(V_e^{P,t})$ is true at the decision point t.

Definition 3.4 *Interval Consistency*. A view $V_e^{P,t}$ is interval consistent if and only if $\phi_{interval}(V_e^{P,t})$ is true at the decision point t.

Interval consistency clearly affords the policy evaluator a high level of confidence in the outcome of the authorization decision. In Sections 3.1 and 3.2, we showed that if certain assumptions could be made about the likelihood of intercredential dependencies and external causes of revocation, then incrementally consistent and internally consistent views can actually become endpoint consistent. Given the above definitions, it should be clear that the following proposition holds.

PROPOSITION 3.5. *An interval consistent view is also endpoint and incrementally consistent, and an endpoint consistent view is also internally consistent.*

One could imagine an extension of interval consistency requiring that all relevant credentials remain valid from the time that they are received until the end of the interaction between the two parties participating in the authorization protocol. That is, if Bob negotiates with GeoTech to gain access to their database (as in Example 1), GeoTech might want to guarantee that it could detect if any of Bob's credentials were revoked after the end of the authorization protocol and consequently prevent Bob from further accessing their database. In Minami and Kotz [2006], the authors propose an authorization system for pervasive computing environments that accomplishes this under the assumption that credential issuers will proactively push revocation information to endpoints in the system. As discussed in Section 2.1, there is no incentive for CAs to maintain the local state necessary to do this in a large open system. In fact, the soundness of algorithms requiring these types of assumptions depends on the reliability with which revocation information is propagated. Enforcement algorithms for the consistency levels discussed in this article will be proven sound without making such assumptions.

4. ALGORITHMS FOR CONSISTENCY ENFORCEMENT

In this section, we discuss the enforcement of the view consistency levels previously presented. We first enumerate the characteristics of an ideal algorithm for consistent view construction and argue that such an algorithm is likely to be impossible to construct in practice. We then discuss two practical algorithms for consistent view construction and use these algorithms to define two extreme points on a multidimensional spectrum of trade-offs affecting view consistency algorithms. We evaluate the costs associated with each of these algorithms and analyze the "distance" of these practical algorithms from the idealized case.

4.1 Comments on the Ideal Case

Each algorithm that we present in this article, and in fact the entire notion of view consistency, is based on the conclusions that can be drawn from the observations of a single entity. As such, the soundness of an algorithm designed to create ϕ-consistent views is only one concern of interest to entities wishing to use that algorithm. Another important goal is quantifying the completeness of this algorithm when compared to an algorithm run by an omniscient entity with complete knowledge of the state of all credentials at all times; we will refer to this as *ideal completeness*. Since entities in any realistic system cannot know the global state of the system at any given time, ideal completeness provides an interesting best case to which the algorithms that we develop can be compared. As we develop the algorithms in this section, we will quantify the shortcomings of these algorithms with respect to ideal completeness. Since incremental consistency is easily implementable, we begin our discussion with an algorithm for constructing internally consistent views.

4.2 Internal Consistency

Algorithm 1 ensures that the views used for authorization policy evaluation are internally consistent. We make the following assumptions in Algorithm 1 (and later algorithms):

—The notation $\leftarrow_r$ denotes random assignment from a set. For example, $s \leftarrow_r \{0, 1\}^m$ assigns to s a random salt value chosen from the set of all length-m binary strings.

—Each entity $e \in \mathcal{E}$ has a set of credentials $C_e = \{c_1, \ldots, c_{n_e}\}$.

—There exists a globally agreed-upon cryptographic hash function $h : \{0,1\}^* \rightarrow \{0,1\}^\ell$ where ℓ is the (fixed) output length of $h(\cdot)$.

—Each entity e chooses a parameter k_e used to hide the number of credentials that she possesses.

—Each entity maintains a hash table, *EntityInfo*, mapping entity names to state information. The function $\mathit{EntityInfo.store} : \mathcal{E} \times (T \setminus \{\perp\}) \times \{0,1\}^m \times 2^{\{0,1\}^\ell} \rightarrow \perp$ stores state information. The function $\mathit{EntityInfo.lookup} : \mathcal{E} \rightarrow (T \setminus \{\perp\}) \times \{0,1\}^m \times 2^{\{0,1\}^\ell}$ retrieves state information.

—Each entity maintains a hash table, *View*, mapping credential identifiers to credential state information. The function $\mathit{View.store} : \mathcal{C} \times \mathcal{S} \rightarrow \perp$ stores credential state information, while $\mathit{View.delete} : \mathcal{C} \rightarrow \perp$ deletes state information.

—The current local time is accessible via the local variable *NOW*.

Algorithm 1 works as follows. At the start of the authorization protocol, each entity calls the INIT method to commit her credentials and a strategically-chosen amount of random noise to the remote party. This allows entities to hide the number of credentials that they possess from their partner in the protocol execution. The amount of noise inserted by each entity is controlled by the parameter k_e, which is strategically chosen by each entity to trade off between privacy and the computational cost of the protocol. Each entity then stores her remote partner's set of committed credentials in the *EntityInfo* hash table. As credentials are received from the remote party during the authorization protocol, the receiver checks to see if the credential was previously committed. If so, the credential state information for this credential is created and stored; if not, the credential is removed from *View*. Should one entity acquire new credentials at runtime, she can recommit her credential set to the remote party by directly using the COMMIT method. If this occurs, the remote party must immediately recheck the semantic validity of each credential stored in the current view and update its associated credential state information (lines 17–23).

This credential recommit process involves fairly high communication overheads for the recipient, as it must contact up to $|\mathit{View}|$ servers to revalidate all potentially relevant credentials. To mitigate denial of service attacks against implementations of this algorithm, entities should require that a recommit message be accompanied by a credential that (i) is relevant at the moment it is received, (ii) was not included in the previous credential set commitment, and (iii) was issued within some fixed window of the time of the last negotiation round. This will ensure that unless parties receive legitimate new credentials, they cannot force excess semantic validity checks. We now highlight several interesting properties of Algorithm 1.

PROPOSITION 4.1. *Any view created using Algorithm 1 is incrementally consistent.*

PROOF. Lines 31–34 of Algorithm 1 ensure that each credential used during the execution of an authorization protocol is valid when it is received. This

Algorithm 1 Internal Consistency

```
{// Initialize a connection with entity $e'$}
**Function** $\textsc{Init}(e' \in \mathcal{E}) = \textsc{Commit}(e')$

{// Commit credentials to entity $e'$}
**Function** $\textsc{Commit}(e' \in \mathcal{E}) =$
$s \leftarrow_r \{0, 1\}^m$ {// create a salt}
$k \leftarrow k_e - |C_e|$ {// need $k$ fake credentials}
**for** $i = 1$ to $k$ **do**
    $r_i \leftarrow_r \{0, 1\}^m$ {// generate fake credentials}
$CC_e \leftarrow \{h(s \mid c_1), \ldots, h(s \mid c_n), h(r_1), \ldots, h(r_k)\}$
Shuffle $CC_e$ randomly
Send $\langle e, s, CC_e \rangle$ to $e'$

{// Receive committed credentials from entity $e'$}
**Function** $\textsc{Rcv}(e' \in \mathcal{E}, s' \in \{0, 1\}^m, CC_{e'} \in 2^{\{0,1\}^\ell}) =$
**if** $EntityInfo.lookup(e') \neq \perp$ **then**
    **for all** $\langle c, r, syn, sem_v, sem_i \rangle \in View$ **do**
        $t \leftarrow NOW$
        **if** $c$ is semantically valid **then**
            $View.store(c, \langle c, r, true, t, sem_i \rangle)$
        **else**
            $View.delete(c)$
$EntityInfo.store(e', \langle NOW, s', CC_{e'} \rangle)$

{// Receive a credential $c$ from entity $e'$}
**Function** $\textsc{Rcv}(e' \in \mathcal{E}, c \in \mathcal{C}) =$
$t \leftarrow NOW$
$\langle rcv, s', CC_{e'} \rangle = EntityInfo.lookup(e')$
**if** $h(s' \mid c) \notin CC_{e'}$ **then**
    Reject $c$
**else if** (($c$ is syntactically valid) and ($\alpha(c) \leq rcv$) and ($c$ is semantically valid)) **then**
    $View.store(c, \langle c, t, true, t, \perp \rangle)$
**else**
    Reject $c$
```

satisfies Definition 3.1 and thus any view created using Algorithm 1 is incrementally consistent. □

PROPOSITION 4.2. *All credentials accepted by Algorithm 1 were held by their bearer at the time of the most recent credential recommit.*

PROOF. For Algorithm 1 to accept some credential c_i from entity e, it must be the case that e committed c_i at the last credential recommit (i.e., $cc_i \in CC_e$). The preimage resistance property of cryptographic hash functions implies that to generate some $cc_i \in CC_e$, e is required to know c_i. This means that either (i) c_i was issued to e prior to the last credential recommit or (ii) e correctly guessed the contents of c_i before it was issued. For case (i), the proposition is true by definition. For case (ii), e must have correctly guessed the signature value that would be placed on c_i by its issuer; this is generally thought to be impossible without knowledge of the issuer's private key. Thus, all credentials accepted by Algorithm 1 were held by their bearer at the time of the most recent credential recommit. □

THEOREM 4.3. *If e's execution of a trust negotiation or distributed proving protocol for target policy P succeeds at time t while using Algorithm 1 to enforce view consistency, then the view $V_e^{P,t}$ is internally consistent.*

PROOF. We proceed by induction on the number of times the COMMIT method is invoked by the remote party. The base case involves one invocation of the COMMIT method; in this case, we will show that all credentials received during the protocol were valid at the time that the credential set was committed. Assume that some credential c such that $\langle c, r, syn_v, syn_i, sem_v, sem_i\rangle \in V_e^{P,t}$ is invalid at the start of the authorization protocol. By Proposition 4.1, c later becomes valid. This contradicts our assumption that once a credential becomes invalid, it cannot again become valid and thus c was valid at the time that the credential set was committed. This implies that all credentials relevant to the satisfaction of P were valid at the time that the credential set was committed. Assume the claim is true for trust negotiation or distributed proving sessions requiring up to $n - 1$ invocations of the COMMIT method. If the trust negotiation or distributed proving session requires n invocations of the COMMIT method, at the time of the n^{th} recommit, lines 17–23 ensure that any previously valid credentials are still valid. By an argument similar to that used in the base case, we know that any credentials accepted after the n^{th} recommit were also valid at the time of the n^{th} recommit. Since all credentials were valid simultaneously at the time of the n^{th} recommit, Definition 3.2 is satisfied and $V_e^{P,t}$ is internally consistent. □

PROPOSITION 4.4. *Algorithm 1 does not disclose credential contents (e.g., credential types or attribute values) to the remote party. Further, if $h(\cdot)$ approximates a random oracle, then no entity can guess the number of credentials held by their communication partner during a given run of the algorithm, nor can they guess the number of new credentials committed during a recommit.*

PROOF. The first property follows from the preimage resistance property of cryptographic hash functions. If $h(\cdot)$ approximates a random oracle, then its output distribution should appear the same regardless of whether its input is a structured credential or a random value. This implies that an adversary cannot determine how many of the committed values correspond to actual credentials versus random noise and therefore the second property holds. To prove the third property, note that because credentials are committed using a different salt for each recommit, unused credentials and random commitments cannot be tracked from recommit to recommit. Note also that newly-acquired credentials replace either a previously unused credential or a random commitment. Clearly if a new credential is used, the remote party can tell that it was in the new set of committed credentials, but not the old set. However, by an argument similar to that used to prove the second property, the adver-

Algorithm 2 Endpoint and Interval Consistency

```
{// Receive a credential $c$ from entity $e'$}
**Function** RCV($e' \in \mathcal{E}, c \in \mathcal{C}$) =
**if** $c$ is syntactically valid **then**
    $View.store(c, \langle c, NOW, true, \perp, \perp \rangle)$
**else**
    Reject $c$

{// Invoked at the end of the access control protocol}
**Function** VALIDATEALL($RelevantCreds \in 2^{\mathcal{C}}$) =
**for all** $\langle c, r, syn, sem_v, sem_i \rangle \in View$ **do**
    **if** $c \in RelevantCreds$ **then**
        $t \leftarrow NOW$
        **if** ($\omega(c) > NOW$) and ($c$ is semantically valid) **then**
            $View.store(c, \langle c, r, true, t, \perp \rangle)$
        **else**
            Fail and report that $c$ is invalid
```

sary cannot tell how many newly-acquired, but unused, credentials may be in a commitment set, so the third property holds. □

Although Theorem 4.3 asserts the soundness of Algorithm 1, this algorithm is not ideally complete as defined in Section 4.1. That is, it is possible for all credentials to be valid simultaneously at the time of the last recommit even if Algorithm 1 fails. Consider the case where Bob commits several credentials to Alice, all of which are valid at the moment the committed credential set is sent to Alice. However, before Alice can verify some credential *c* that was committed by Bob, *c*'s issuing CA revokes the credential. Alice thus cannot tell that *c* was valid at the time that the credential set was committed, though an omniscient entity could. In Section 5.4, we propose an online credential status protocol that allows Algorithm 1 to more closely approximate ideal completeness.

4.3 Endpoint and Interval Consistency

Algorithm 2 guarantees that all executions of an authorization protocol that succeed do so using interval consistent views. In general, the strategy adopted by this algorithm is similar to that taken in optimistic concurrency control algorithms for transaction management. That is, credentials are syntactically validated as they arrive, as this can be done without external interaction, but are assumed to be semantically valid. When a decision point is reached, the VALIDATEALL method is invoked to check the semantic validity of each relevant credential in the view and terminate the protocol if any credentials are found to be invalid. Because *e*'s strategy has reached a decision point, it will have the clearest idea yet as to which submitted credentials are actually relevant. If one of these credentials is invalid, however, *e*'s strategy can continue to search for another set that satisfies the policy; this new set can then be checked for validity, and so on. If only endpoint consistent views are required, then both the semantic and syntactic validity checks can be delayed until the VALIDATEALL method.

THEOREM 4.5. *If an execution of a trust negotiation or distributed proving protocol for a target policy P succeeds at time t while using Algorithm 2 to enforce view consistency, then the view $V_e^{P,t}$ is interval, endpoint, internally and incrementally consistent.*

PROOF. Line 3 ensures that for each $\langle c, r, syn_v, syn_i, sem_v, sem_i \rangle \in V_e^{P,t}$, the credential c was syntactically valid at time $t_i \leq r$. Line 13 ensures that each c_i was semantically valid at some time $t_i' \geq end(V_e^{P,t})$ and thus $V_e^{P,t}$ is interval consistent by Corollary 2.6. It is therefore endpoint, internally, and incrementally consistent by Proposition 3.5. □

Although Algorithm 2 is sound (by Theorem 4.5) it is not ideally complete. Since the VALIDATEALL method takes some finite, but non-instantaneous, amount of time to check the semantic validity of each c_i whose state is stored in V, it is entirely possible that each c_i was valid at $end(V)$, but one such credential was revoked before its semantic validity could be checked by the algorithm. An omniscient entity could detect this event, even though it would go undetected by Algorithm 2. The well-known limitations of causal orderings and virtual clocks [Lamport 1978; Cheriton and Skeen 1993] lead us to the following assertion regarding the ideal completeness of endpoint and interval consistency algorithms.

THEOREM 4.6. *Sound and ideally-complete endpoint and interval consistency algorithms can exist if and only if the entity e constructing the view $V_e^{P,t}$ can synchronize clocks with the issuer of each credential c whose state information is stored in $V_e^{P,t}$.*

PROOF. We first show that clock synchronization is a necessary condition for defining sound and ideally-complete endpoint and interval consistency algorithms. The definitions of ϕ_{end} and $\phi_{interval}$ require that each credential c whose state information is stored in $V_e^{P,t}$ be semantically valid at the exact time $end(V_e^{P,t})$, where $end(V_e^{P,t})$ is defined by Equation 4 as the moment that the last credential relevant to the satisfaction of P was received by e. This implies that e and each CA that issued a credential c_i whose state information is stored in $V_e^{P,t}$ must be able to agree on the precise instant $end(V_e^{P,t})$ to correctly check the validity of each c_i at $end(V_e^{P,t})$. Since $end(V_e^{P,t})$ depends on delays in both the network and e's local processing queues, causal relationships cannot be used to facilitate this agreement and thus e must be able to synchronize clocks with each CA that issued a credential whose state information is stored in $V_e^{P,t}$.

We demonstrate that the ability to synchronize clocks with credential issuers is a sufficient condition for defining sound and ideally-complete endpoint and interval consistent views by sketching a protocol that accomplishes this task. Figure 7 illustrates an online credential status protocol that tells the requester, e, not only whether the credential c whose status was requested is still valid (via the *valid* field), but also the most recent instant in time the creden-

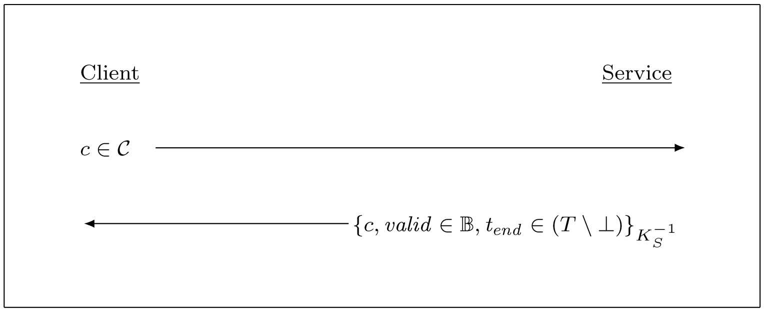
Client
Service
$c \in \mathcal{C}$
$\{c, valid \in \mathbb{B}, t_{end} \in (T \setminus \bot)\}_{K_S^{-1}}$

Fig. 7. An online credential status protocol leveraging synchronized clocks.

tial was observed valid by the CA (via the t_{end} field). In the event that *valid* is false, then t_{end} represents the time at which c expired or was revoked; if *valid* is true, then t_{end} is the time at which the CA responded to e's request.

Note that clock synchronization allows t_{end} to be translated to be relative to e's local clock; this field can be combined with a similarly translated value of $\alpha(c)$ to give e an accurate view of c's validity interval. If the protocol presented in Figure 7 is used to make the semantic validity check that occurs on line 13 of Algorithm 2, e can accurately establish the concurrent validity of all credentials that make up $V_e^{P,t}$. This is in contrast to a version of Algorithm 2 relying on a certificate validation protocol like OCSP. In this case, the algorithm can fail to recognize an endpoint or interval consistent view if some credential c is valid at $end(V_e^{P,t})$ but becomes revoked prior to having its validity checked. This can occur because no causal relationship can be established between $end(V_e^{P,t})$ and the validity of c; the ability to synchronize clocks removes the need for this type of causal relationship. Since the version of Algorithm 2 using the protocol presented in Figure 7 is still sound (by Theorem 4.5), this shows that clock synchronization is a sufficient condition for constructing a sound and ideally-complete endpoint or interval consistency enforcement algorithm. □

4.4 Trade-offs in Consistency Enforcement

In examining Algorithms 1 and 2, a clear trade-off emerges. By deferring semantic validation checks until the end of the protocol, Algorithm 2 reduces the work for the verifier by allowing her to semantically validate only the credentials that were ultimately determined to be relevant to the satisfaction of the policy. This reduction in work comes at a price, however. In the case that the policy being satisfied uses guard conditions to protect the disclosure of more sensitive portions of the policy (e.g., as in Bonatti and Samarati [2000] and Li et al. [2005]) optimistically assuming that credentials are semantically valid could leak sensitive policy information to unauthorized viewers. To correct this problem, each set of guard conditions must be viewed as a subnegotiation in its own right, so that the semantic validity of the credentials satisfying the guard conditions is checked before access is granted to the remaining policy.

Alternatively, Algorithm 1 can be modified to call the VALIDATEALL method at its conclusion. However, Algorithm 1 incurs much higher overheads for the verifier, as each credential received must be validated throughout the protocol, as its relevance cannot be fully determined until the end of the protocol.

These algorithms are two extreme points on the spectrum of possible consistency enforcement algorithms. In some cases, an entity may prefer to aggressively monitor the validity of some credentials received over the course of the authorization protocol, while deferring checks on other credentials. For instance, for the policy $P = c_1 \wedge (c_2 \vee c_3)$, it is clear that c_1 is relevant to the satisfaction of P. Thus c_1 could be monitored more aggressively (using a scheme like that in Algorithm 1), while checks on the validity of credentials c_2 and c_3 could be delayed until the end of the protocol. Designing consistency enforcement algorithms that balance this trade-off between relevance, work for the verifier, and information leakage will be an interesting challenge.

5. DISCUSSION

In this section, we discuss several interesting facets of view consistency. In particular, we show that the algorithms presented in this article satisfy the requirements presented in Section 2.3, consider the effects of an unstable environment on view consistency, and introduce the notion of strategic algorithms for view consistency enforcement. We then propose a novel online credential status protocol that allows Algorithm 1 to very closely approach ideal completeness, comment on the effects of poorly synchronized CA clocks, and discuss practical considerations relating to the deployment of the algorithms presented in this article.

5.1 Requirements Revisited

In Section 2.3 we presented three requirements that view consistency algorithms should satisfy: loose clock synchronization, minimal cooperation, and minimal impact on existing protocols. Each algorithm presented in this article relies only on its local perception of time and causal event orderings; no synchronization with external sources is necessary. Further, only a small amount of cooperation between entities is required for these algorithms to function correctly. Specifically, in Algorithm 1, only the two parties engaged in the authorization protocol need to cooperate to form a consistent view. The only way that the remote party can fail to cooperate in these algorithms is to incorrectly commit her credential values; this failure can only deny her access to the requested resource. Algorithm 2 requires no cooperation between entities in the system to succeed. Lastly, the algorithms presented in this article have virtually no impact on existing trust negotiation and distributed proving protocols, as they were designed to wrap the functionality already provided by existing protocols and systems. By disabling credential verification in existing systems and using wrapper code that implements the consistency checking algorithms presented in this article, existing systems can enforce stronger levels of view consistency.

5.2 Dynamic Environments

In context-rich environments like smart buildings and grid computing systems, it is entirely possible for authorization policies to be predicated on the state of the surrounding environment. For instance, authorization policies may consider the time of day or the occupancy status of a room. A malicious client can attempt to alter the state of their surrounding environment in unexpected ways to twist the outcome of an authorization protocol.

The environmental inputs to an authorization protocol can consist of either certified environmental information collected by the client (or some agent acting on his behalf) or observations made by the resource provider. In the event that only certified environmental information is used, then the endpoint and interval consistency algorithms presented in this article can ensure that all environmental assertions remain true throughout the duration of the authorization protocol. However, ensuring that observational data regarding system context does not become invalidated is a more difficult task. The resource provider must either continuously monitor the pertinent state information or register to be alerted should its value change. Periodically checking the state is insufficient, as fluctuations of the value between checks cannot be detected. If the resource provider has the capability to register such alerts, then this mechanism combined with one of the algorithms presented in this article can ensure that the consistency of their view can be protected from the effects of unstable environmental conditions that are either naturally occurring or maliciously induced.

5.3 Strategic Algorithm Design

Trust negotiation and distributed proving are dynamic processes, the properties of which depend on the strategies or tactics adopted by their participants [Bauer et al. 2005; Winsborough and Li 2006; Yu et al. 2003]. Similarly, the level of view consistency required by a given entity is to some extent also a strategic decision (this is a further extension of the trade-off noted in Section 4.4). The consistency enforcement algorithms presented in this article were designed to enforce various safety properties, and thus our algorithms focused on satisfying only these criteria. However, consistency may not always be the only concern for some resource providers. Rather, they may wish to enforce some level of consistency but require algorithms with stronger guarantees regarding the availability of their services (i.e., they require an algorithm that closely approaches ideal completeness) or privacy preservation than those provided by the algorithms in this article.

For instance, recall that Algorithm 1 allows an entity Alice to hide her credentials in a set of credentials and fake commitments of size k_e. To do this, however, requires that she compute and disclose the results of k_e hashes; the overhead of this process quickly becomes burdensome as k_e increases. As a more efficient option, we can use Merkle trees [Merkle 1979] to allow Alice to hide her n credentials in a set of $k_e = 2^s$ fake credentials with minimal overheads. Specifically, Alice need only compute and disclose a single commitment value to hide her n credentials and her negotiation partner need only compute

s hashes to determine whether a given credential is contained in a particular commitment. We now briefly present this scheme, describe the upper bound on its running time, and compare it to the commitment scheme presented as part of Algorithm 1.

If Alice wishes to create a commitment hiding her n credentials in a set of $k_e = 2^s$ possible credentials, she first assigns each of her credentials a random identifier from the set $\{0, 1\}^s$ and then creates a binary tree with 2^s leaves, where each leaf corresponds to exactly one identifier in the set $\{0, 1\}^s$. Alice then hashes each of her credentials and places each credential's hash value at the leaf of the tree corresponding to its identifier. Each unused subtree of the binary tree is then removed and replaced with a random string from $\{0, 1\}^\ell$ henceforth referred to as a *fake subtree*; recall from Section 4.2 that ℓ is the output bit length of the agreed-upon hash function, $h(\cdot)$. At this point, Alice computes the Merkle hash of this modified binary tree and discloses this single ℓ-bit value as her commitment.

PROPOSITION 5.1. *In the worst case, the Merkle tree commitment algorithm requires $O(ns)$ hash computations to produce a commitment value for n credentials in a set of $k_e = 2^s$ possible credentials.*

PROOF. Note that the Merkle commitment algorithm achieves its worst running time when the number of fake subtrees is maximized, as this maximizes the number of hash operations required to combine all real credentials and fake subtrees; in practice, the number of fake subtrees is maximized when the identifiers assigned to an entity's actual credentials are uniformly distributed across the identifier space $\{0, 1\}^s$. For ease of exposition, assume that the number of credentials held by a given entity is a power of 2. In this case, the resulting *pruned* binary tree constructed by the commitment algorithm will consist of n tendrils of length $\log k_e - \log n = s - \log n$ containing $s - \log n$ fake subtrees and the hash of one real credential; these tendrils join the leaves of a complete binary tree of depth $\log n$ (see Figure 8). Computing the Merkle hash of this pruned tree then requires $n(s+1) - n\log n - 1$ hash operations: $s - \log n$ hashes for each of the n tendrils and $n - 1$ hashes to combine these n tendrils when hashing the complete binary tree of depth $\log n$. □

For Alice to enable her partner in the authorization protocol to verify that a particular credential c is incorporated in her commitment value, she discloses the hash values of the s nodes along the path from c to the root of the Merkle tree and the hash values representing the s subtrees connected to this path. Her partner in the protocol can then recompute the value of the root of the Merkle tree, thereby verifying that c is incorporated in this tree; this process requires s hash computations.

PROPOSITION 5.2. *The Merkle tree commitment algorithm does not disclose credential contents (e.g., credential types or attribute values) to the remote party. Further, if $h(\cdot)$ approximates a random oracle, then no entity can guess the number of credentials held by their commitment partner during a given run of the algorithm, nor can they guess the number of new credentials committed during a recommit.*

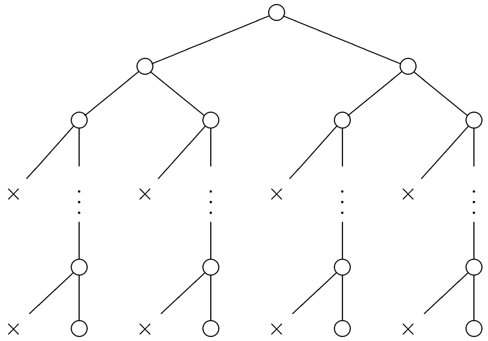

Fig. 8. An example worst-case pruned binary tree for an instance of the Merkle commitment scheme in which an entity is committing four real credentials. Fake subtrees are denoted with an × character.

Table II. Required Numbers of Hashes and Corresponding Hash Computation Times for Various Configurations of the Merkle Commitment Algorithm

$\mathbf{k_e}$	**Number of actual credentials**				
	16	32	64	128	256
2^{16}	207 (0.5 ms)	383 (0.6 ms)	703 (1.8 ms)	1279 (3.4 ms)	2303 (6.3 ms)
2^{32}	463 (1.2 ms)	895 (2.8 ms)	1727 (4.7 ms)	3327 (8.9 ms)	6399 (16.8 ms)
2^{64}	975 (2.8 ms)	1919 (5.2 ms)	3775 (10.3 ms)	7423 (19.8 ms)	14591 (38.9 ms)
2^{128}	1999 (5.4 ms)	3967 (10.5 ms)	7871 (20.9 ms)	15615 (45.0 ms)	30975 (83.0 ms)

PROOF. As in the proof of Proposition 4.4, the preimage resistance property of $h(\cdot)$ and the fact that $h(\cdot)$ approximates a random oracle prevent the remote party from distinguishing between real and fake leaves of the Merkle tree. By induction, this prevents the remote party from distinguishing between real and fake subtrees of the Merkle tree. This implies that the remote party cannot determine the number of real credentials committed in a particular value, aside from knowing that it is at least the number of credentials disclosed and verified during the execution of the protocol and less than or equal to 2^s. The third property follows directly from this fact. □

Table II contains the number of hash operations required to generate Merkle commitments for between 16 and 256 actual credentials hidden in sets containing between 2^{16} and 2^{128} potential credentials, along with the times required to compute these numbers of hashes. Timings were calculated using a Java implementation executed on a 2.5 GHz Pentium 4 with 512MB RAM running Linux. All times reported are averages over 10 repeated trials. The resulting running times indicate that entities can easily commit their credentials into extremely large anonymity sets with only minimal computational and data transmission overheads, compared to those that would be imposed by the commitment scheme used in Algorithm 1. When combined with Propositions 5.1 and 5.2, this shows that changing only the commitment scheme used by Algorithm 1 allows us to tune both the performance and privacy guarantees of the algorithm without affecting the consistency property that it enforces. This

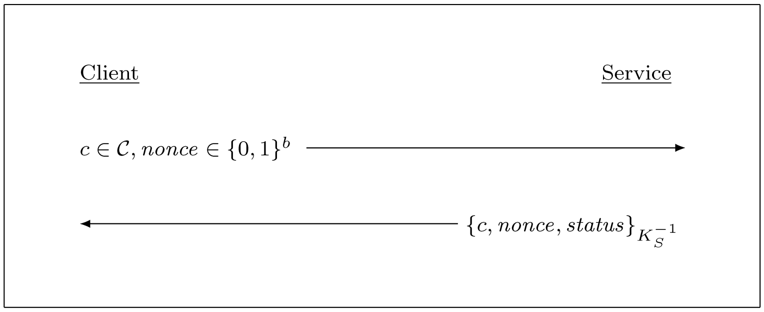
Client
Service
$c \in \mathcal{C}, nonce \in \{0,1\}^b$
$\{c, nonce, status\}_{K_S^{-1}}$

Fig. 9. An online credential status protocol.

suggests that further analysis of these types of strategic trade-offs in view consistency algorithms may be an interesting area of future research.

5.4 Towards Completeness for Internal Consistency Algorithms

We now propose an online credential status verification protocol that, when used in conjunction with Algorithm 1, allows the modified Algorithm 1 to more closely approach ideal completeness. Figure 9 illustrates this two-message protocol. In this protocol, a client provides the verification service with a credential whose status she wishes to verify and a nonce value whose length is chosen by the client. The service then determines the current validity status of the provided credential and returns the credential, the nonce, and the current status of the credential signed with its private key, K_S^{-1}, whose public counterpart, K_S, is assumed to be well-known.

Recall that Algorithm 1's shortcomings with respect to ideal completeness arise when all of Bob's credentials are valid when they are committed to Alice, but some credential is revoked before Alice validates it. Now, assume that each CA runs the online credential status verification service implementing the protocol presented in Figure 9. If Alice chooses a random nonce and sends it to Bob prior to Bob committing his credential set to Alice, Bob can obtain certified validity statements for each of his credentials from their respective issuing CAs, each of which includes Alice's nonce. Bob can then commit these validity statements along with each of his credentials to Alice. As Bob discloses a credential to Alice during the authorization protocol, he must also disclose its associated certified validity statement to Alice. Figure 10 illustrates how this protocol extends the commitment phase of Algorithm 1 to become a four-step process. Note that although only the initial commitment of Algorithm 1 is shown in Figure 10, this protocol can be used for all commitments made during an execution of the algorithm. Alice can now verify that the credential was valid at the time that it was committed by Bob.

PROPOSITION 5.3. *If a credential c and its associated certified validity statement $cvs = \{c, nonce, true\}_{K_S^{-1}}$ are contained in the commitment set received by Alice, then c was valid at the time that Alice disclosed her b-bit nonce to Bob with probability $1 - 2^{-b}$, provided that Alice chose her nonce value at random.*

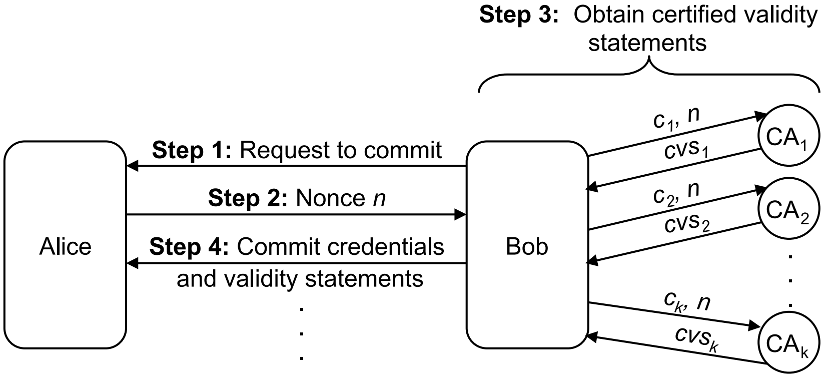
Step 3: Obtain certified validity statements
Step 1: Request to commit
Step 2: Nonce n
Step 4: Commit credentials and validity statements
Alice
Bob
c1, n
cvs1
c2, n
cvs2
ck, n
cvsk
CA1
CA2
CAk

Fig. 10. An illustration of how the protocol presented in Figure 9 can be integrated with the commitment phase of Algorithm 1.

PROOF. Assume that Bob obtained *cvs* prior to the time that Alice disclosed *nonce*. This implies that Bob correctly guessed *nonce*, which he can do only with probability 2^{-b} if Alice chose *nonce* at random. Thus, with probability $1 - 2^{-b}$, Bob obtained *cvs* after Alice disclosed *nonce*. As long as Alice ensures that $\alpha(c)$ is less than or equal to the time that she sent Bob *nonce*, then she can conclude that c was valid at the time that she disclosed *nonce* to Bob (by Proposition 2.5). □

The above proposition allows Alice to conclude that all credentials used during the authorization protocol were valid at the time of the most recent recommit, provided that she chooses a new nonce for each recommit. This credential status protocol allows a modified version of Algorithm 1 to more closely approximate ideal completeness. An added benefit of this protocol is that it allows Alice to shift the responsibility of verifying the semantic validity of Bob's credentials to Bob; if Alice is a very busy resource provider, this could allow her to increase the number of trust negotiation sessions that she can complete per unit time. However, this modified Algorithm 1 is still incomplete, as each of Bob's credentials may be valid when he receives Alice's nonce, but one of them might be revoked prior to his obtaining a certified credential validity statement from its issuing CA. This is similar to the problem discussed in Section 4.3 in which Algorithm 2 could fail because validating all relevant credentials takes a nonzero amount of time. As in that case, it is unlikely that ideal completeness could be reached without the assumption of synchronized clocks.

5.5 A Note of Caution Regarding CA Clock Skew

The algorithms in this article assume that the times $\alpha(c)$ and $\omega(c)$ are interpreted relative to the local clock, as is done in commodity software like Web browsers. That is, if the local clock indicates that $\omega(c)$ not yet passed, then c is accepted as syntactically valid. While in many cases this is a safe assumption to make, especially if online semantic validity checks are also made, it can in some cases lead to troubles if CA clocks are poorly synchronized. For example, consider the case in which an entity receives credentials c_1 (issued by CA 1 and expiring at time t_1) and c_2 (preissued by CA 2 and becoming valid at time $t_2 \leq t_1$) as part of an authorization protocol. Based on the local interpretation of t_1 and t_2, the validity period of these credentials overlaps. However, if the clock at CA 2 is slower than the clock at CA 1 by at least $t_1 - t_2$, then despite appearing to overlap, the validity intervals of c_1 and c_2 never actually overlap.

Fortunately, the use of time synchronization protocols such as NTP [Mills 1992] by service providers reduces the likelihood of this type of error randomly occurring between unrelated credentials. It is vital that CAs closely synchronize their clocks if they issue mutually-exclusive certificates, so that no misleading apparent overlaps can occur. Such apparent overlaps are not introduced by the algorithms developed in this article, but rather by the widespread notion of using a local interpretation of certificate expiration times. Fortunately, there is no way for an attacker to exploit this type of error without altering the clock of at least one CA before it issues a certificate that is subsequently used in the negotiation that the attacker wishes to disrupt.

5.6 Deployment Considerations

It is clear that enforcing the levels of consistency discussed in this article comes at the expense of increased computational and communication complexity. For example, Algorithm 1 imposes computational and communication overheads associated with creating and transmitting credential commitments, as well as communication overheads resulting from semantic validity checks. Recall from Section 5.3 that the costs of committing a set of credentials can be greatly minimized through the use of Merkle trees, so the total overhead associated with consistency enforcement is dominated by the costs of semantic validity checks on the credentials exchanged during the protocol. Since the semantic validity checks executed during any one round of a negotiation can be carried out by accessing the appropriate CAs or revocation servers in parallel, the running time overhead of Algorithm 1 scales as approximately $O(RA)$, where R is the number of rounds in the negotiation and A is the average time required for a single semantic validity check. Similarly, Algorithm 2 requires that the semantic validity of each credential be checked at the end of the protocol and thus has running time overheads that scale as approximately $O(A)$, as these checks can also be parallelized.

Unfortunately, carrying out these types of semantic validity checks cannot be avoided; without them, we cannot conclude whether a single credential is valid at a given point in time, much less whether a view relating multiple credentials is consistent. The actual overheads imposed by Algorithms 1 and 2 depend on the protocol used to check the semantic validity of a given credential. For example, Zhou et al. [2002] show that querying the revocation status of a credential using a four-node deployment of the COCA system whose nodes are spread across the United States and Europe takes an average of $A = 2.27$ seconds. Given that COCA process groups are designed to tolerate Byzantine failures and use threshold cryptography and proactive secret sharing to ensure correct responses in the face of node compromises, this is not surprising. However, this could lead to very high overheads if used in conjunction with Algorithm 1. On the opposite end of the spectrum are simpler credential status checking protocols, such as OCSP and the protocols presented in Figures 7 and 9. In these cases, A would be approximately the time required for one round trip between the verifier and CA.

The consistency levels defined in this article have been adapted [Lee et al. 2007] to fit within the Minami-Kotz distributed proof system [Minami and Kotz 2005]. A status checking protocol similar to that presented in Figure 9 was used to verify fact validity statuses and resulted in overheads of less than 30 percent during the proof construction and validation process. In these experiments, fact status checks were not parallelized; this overhead could be further reduced through parallelization. While there is not a direct mapping between the Minami-Kotz distributed proof protocol and the trust negotiation protocols discussed in this article, these low overheads are promising. When coupled with the asymptotic analysis presented above, they suggest that the overheads of semantic validity checks should be tolerable. This is particularly true if the semantic validity check process can be safely offloaded to client processes as discussed in Section 5.4, since heavily-loaded servers will then be freed from further burdens.

6. RELATED WORK

Informally, a safety property of a distributed algorithm is a guarantee that some (presumably bad) situation will not arise [Lamport 1977]. Previous works regarding the safety of trust negotiation systems have focused on ensuring that private information will not be inadvertently leaked during the trust negotiation process. Yu et al. [2003] first defined the notion of “safe disclosure sequences.” Informally, they consider a trust negotiation safe if each resource disclosed during the negotiation was “unlocked” (i.e., its authorization policy was satisfied) at the time that it was disclosed. Winsborough and Li [2006] note that under this notion of safety, private information that is not explicitly revealed during a trust negotiation can still be inferred based on the way that an entity carries out the negotiation. They propose several more refined notions of safety for trust negotiation protocols based on the concept of indistinguishability, each of which gives users stronger guarantees regarding the amount of private information leaked during the negotiation. Irwin and Yu [2005] propose another definition of safety based on the idea of information gain. Our work is orthogonal to these previous works in that we are concerned with ensuring that access to system resources will not be granted based on an inconsistent interpretation of the underlying system state. It would be prudent for system designers to consider both types of safety.

Another area of closely related work is that of concurrency control and consistency enforcement in distributed systems, distributed databases, and distributed shared memory. Each of these areas has a rich body of literature, surveys of which can be found in Tanenbaum and van Steen [2002], Cellary et al. [1988], and Adve and Gharachorloo [1996], respectively. In general, these problem domains assume that multiple entities will be updating values stored at multiple locations within the system and as such, maintaining data consistency is of concern to everyone. Therefore, solutions to the transaction management problem in these domains typically involve the cooperation of multiple entities, as every entity has incentive to cooperate. However, as was discussed in Sections 1 and 2.1, groups of entities have no incentive to cooperate in solving the view consistency problem for trust negotiation and distributed proving since this problem is of concern only to a particular resource provider evaluating a particular policy. Therefore, the solutions developed in the distributed systems, distributed databases, and distributed shared memory literature are unsuitable for our problem domain; the solutions that we develop in this article require only the cooperation of, at most, the two parties participating in the authorization protocol.

A final area of related work is the collection of system state snapshots in distributed systems. Collecting consistent snapshots that can be used to evaluate stable predicates over the system state is a well-known problem, to which an elegant solution was presented by Chandy and Lamport [1985]. This algorithm is not directly applicable to the problem addressed in this article, however, due to the unstable nature of credential statuses. There exist algorithms for collecting distributed state snapshots that can be used to evaluate unstable predicates (for a survey, see Babaoğlu and Marzullo [1993]), though these algorithms have very high overheads and make unreasonable assumptions about process cooperation for our problem domain.

7. CONCLUSIONS AND FUTURE WORK

In this article, we presented the notion of view consistency in trust negotiation and distributed proving authorization systems. We showed that failing to consider the consistency of the system views used during executions of these protocols can cause a marked decrease in the safety of the decisions made by the underlying authorization system. We then defined the incremental, internal, endpoint, and interval consistency levels and demonstrated algorithms to attain these consistency levels in practice. We proved the soundness of each of these algorithms and commented on their completeness when compared to an ideal algorithm run by an omniscient entity. These algorithms require at most the cooperation of the two parties involved in the authorization process; should any entity not cooperate, the algorithms will fail rather than violate the consistency conditions that they were designed to enforce. We then explored the notion of strategic design trade-offs for consistency enforcement algorithms. This led us to propose several modifications to the algorithms presented in Section 4 that enhance the privacy-preservation properties of these algorithms or their closeness to ideal completeness without altering the consistency constraints that they enforce.

We are currently pursuing one important area of future work involving generalizing the consistency enforcement algorithms presented in this article to work not only with attribute or environmental state information encoded in certificates, but also with more general sources of trustworthy information. As an example, consider the distributed proof construction algorithm presented by Minami and Kotz [2005]. In this article, the authors present a distributed proof construction algorithm for pervasive computing environments in which proofs are justified by atoms provided by nodes in the system. As it would be unreasonable to expect each node in the system to run its own CA, these atoms are simply signed using an RSA private key whose corresponding public key is associated with the node. One could imagine this model also being adopted in sensor networks, perhaps using keyed MACs rather than digital signatures. These types of systems present some of the same challenges described in Sections 1 and 2 of this article—namely that entities wish to use decentralized information sources that have no incentive to cooperate with one another to make decisions—and could thus benefit from the types of consistency enforcement mechanisms proposed in Section 4. Unfortunately, the mechanisms presented in this article rely heavily on the semantics of certificate revocation and thus cannot be used directly in systems using more general forms of trusted information. We are in the process of designing more general consistency enforcement strategies that are applicable to a wider range of systems using decentralized knowledge to make decisions in adversarial environments. Initial results in this area can be found in Lee et al. [2007].

Additionally, there are several other areas of interesting future work relating to view consistency. As alluded to in Section 5.3, the design of consistency enforcement algorithms that make a variety of trade-offs regarding safety, availability, and privacy-preservation properties could prove to be a fruitful area of investigation. Given the autonomous nature of the entities participating in trust negotiation and distributed proving authorization protocols, it would be beneficial to explore the notion of interoperable families of algorithms for consistency enforcement (as was done in Yu et al. [2003] for trust negotiation strategies). This would allow each entity to acquire the consistency level she requires without placing unnecessary constraints on her communication partners. Another area of future work involves the development of consistent views shared by several entities in the system. Given the falling costs associated with fine-grained clock synchronization via technologies such as GPS and an increased interest in distributed authorization, interesting notions of view consistency are likely to emerge from a study of this topic.

ACKNOWLEDGMENT

The authors wish to thank an anonymous reviewer who suggested the Merkle commitment scheme explored in Section 5.3.

REFERENCES

ADVE, S. V. AND GHARACHORLOO, K. 1996. Shared memory consistency models: A tutorial. *IEEE Comput.* 66–76.

BABAOĞLU, O. AND MARZULLO, K. 1993. Consistent global states of distributed systems: Fundamental concepts and mechanisms. In *Distributed Systems*, S. J. Mullender, ed. Addison-Wesley, 55–96. Also available as University of Bologna Tech. rep. UBLCS-93-1 at http://www.cs.unibo.it/pub/TR/UBLCS/1993/93-01.ps.gz.

BAUER, L., GARRISS, S., AND REITER, M. K. 2005. Distributed proving in access-control systems. In *Proceedings of the IEEE Symposium on Security and Privacy (SSP'05)*. 81–95.

BECKER, M. Y. AND SEWELL, P. 2004. Cassandra: Distributed access control policies with tunable expressiveness. In *Proceedings of the 5th IEEE International Workshop on Policies for Distributed Systems and Networks (NDSS'04)*. 159–168.

BERTINO, E., FERRARI, E., AND SQUICCIARINI, A. C. 2004. Trust-X: A peer-to-peer framework for trust establishment. *IEEE Trans. Knowl. Data Eng. 16*, 7, 827–842.

BONATTI, P. AND SAMARATI, P. 2000. Regulating service access and information release on the web. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS'00)*. 134–143.

CELLARY, W., GELENBE, E., AND MORZY, T. 1988. *Concurrency Control in Distributed Database Systems*. Elsevier Science Publishing.

CHANDY, K. M. AND LAMPORT, L. 1985. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst. 3*, 1, 63–75.

CHERITON, D. R. AND SKEEN, D. 1993. Understanding the limitations of causally and totally ordered communication. In *Proceedings of the ACM Symposium on Operating Systems Prinicíples (SOSP'93)*. 44–57.

HOUSELY, R., FORD, W., POLK, W., AND SOLO, D. 1999. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. IETF Request for Comments RFC-2459.

IRWIN, K. AND YU, T. 2005. Preventing attribute information leakage in automated trust negotiation. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*. 36–45.

KOSHUTANSKI, H. AND MASSACCI, F. 2005. Interactive credential negotiation for stateful business processes. In *Proceedings of the 3rd International Conference on Trust Management (iTrust'05)*. 257–273.

LAMPORT, L. 1977. Proving the correctness of multiprocess programs. *IEEE Trans. Softw. Eng. SE-3*, 2, 125–143.

LAMPORT, L. 1978. Time, clocks, and the ordering of events in a distributed system. *Comm. ACM 21*, 7, 558–565.

LEE, A. J., MINAMI, K., AND WINSLETT, M. 2007. Lightweight consistency enforcement schemes for distributed proofs with hidden subtrees. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT'07)*. 101–110.

LEE, A. J. AND WINSLETT, M. 2006. Safety and consistency in policy-based authorization systems. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS'06)*. 124–133.

LI, J., LI, N., AND WINSBOROUGH, W. H. 2005. Automated trust negotiation using cryptographic credentials. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*. 46–57.

LI, N. AND MITCHELL, J. 2003. RT: A role-based trust-management framework. In *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX'03)*. 201–213.

MERKLE, R. C. 1979. Secrecy, authentication, and public key systems. Ph.D. thesis, Stanford University.

MILLS, D. L. 1992. Network Time Protocol (Version 3) Specification, Implementation and Analysis. IETF Request for Comments RFC-1305.

MINAMI, K. AND KOTZ, D. 2005. Secure context-sensitive authorization. *J. Perv. Mob. Comput. 1*, 1, 123–156.

MINAMI, K. AND KOTZ, D. 2006. Scalability in a secure distributed proof system. In *Proceedings of the 4th International Conference on Pervasive Computing (PERVASIVE'06)*. 220–237.

MYERS, M., ANKNEY, R., MALPANI, A., GLAPERIN, S., AND ADAMS, C. 1999. X.509 Internet public key infrastructure online certificate status protocol - OCSP. IETF RFC 2560.

TANENBAUM, A. S. AND VAN STEEN, M. 2002. Distributed systems: Principles and Paradigms. Prentice Hall, Upper Saddle River, NJ.

WINSBOROUGH, W. H. AND LI, N. 2002. Towards practical automated trust negotiation. In *Proceedings of the 3rd IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*. 92–103.

WINSBOROUGH, W. H. AND LI, N. 2006. Safety in automated trust negotiation. *ACM Trans. Inf. Syst. Secur. 9*, 3, 352–390.

WINSLETT, M., YU, T., SEAMONS, K. E., HESS, A., JACOBSON, J., JARVIS, R., SMITH, B., AND YU, L. 2002. Negotiating trust on the web. *IEEE Internet Comput. 6*, 6, 30–37.

WINSLETT, M., ZHANG, C., AND BONATTI, P. A. 2005. PeerAccess: A logic for distributed authorization. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*. 168–179.

YU, T., WINSLETT, M., AND SEAMONS, K. E. 2003. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Trans. Inf. Syst. Secur. 6*, 1, 1–42.

ZHOU, L., SCHNEIDER, F. B., AND VAN RENESSE, R. 2002. COCA: A secure distributed online certification authority. *ACM Trans. Comput. Syst. 20*, 4, 329–368.

Received January 2007; revised July 2007; accepted August 2007