Safety and Consistency in Policy-Based Authorization Systems (Extended Version)[*]

Adam J. Lee and Marianne Winslett
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{adamlee, winslett}@cs.uiuc.edu

Abstract

In trust negotiation and other distributed proving systems, networked entities cooperate to form proofs that are justified by collections of certified attributes. These attributes may be obtained through interactions with any number of external entities and are collected and validated over an extended period of time. Though these collections of credentials in some ways resemble partial system snapshots, these systems currently lack the notion of a consistent global state in which the satisfaction of authorization policies should be checked. In this paper, 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 which provide stronger safety guarantees. We provide algorithms that allow each of these refined notions of consistency to be attained in practice with minimal overheads.

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 [4, 5, 12, 14, 15, 21, 23, 25] and distributed proving [3, 18, 24]. 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 paper we

[*]This paper is an extended version of the paper "Safety and Consistency in Policy-Based Authorization Systems" that appears in the Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS 2006).

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 pre-ordained expiration time) then policy evaluation can be reduced to the problem of stable predicate evaluation on distributed snapshots [8]. 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. This is especially worrisome in trust negotiation and distributed proving, 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. 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 trial access to the database, provided that their company authorizes them to make purchases of over $10,000 (the cost of a department subscription to the database). 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 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 leads to the permission of an undesirable access.

Example 2. Alice is a PhD 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 US 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 non-instantaneous 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 [20], distributed databases [7], and distributed shared memory [1], it 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 paper 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 paper, 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, 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, though due to the lack of pre-existing trust relationships in the system, traditional identity-based solutions to this problem cannot be used. As such, trust negotiation or distributed proving approaches to authorization will be used.

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 [14, 25]; 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 [3, 18, 24]) 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 paper, we will denote the set of all credentials by $\mathcal{C}$. 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 [10], $\alpha(c)$ 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. 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) [19] or by an online CA such as COCA [26].

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 paper, we are concerned with two types of credential validity: syntactic and semantic.

Definition 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 (Semantic Validity). *A credential c issued at time t_i is* semantically valid *at time t if an online method of verifying c's status indicates that c was not revoked at time t' and $t_i \leq t \leq t'$.*

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 degenerate 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 paper.

Definition 3 (Validity). *A credential c is* valid *at time t if it is syntactically and semantically valid at time t.*

Proposition 1. *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. *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 1. *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 4 (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 $s^e_c = \langle c, r, syn, sem_v, sem_i \rangle \in \mathcal{C} \times (T \setminus \{\perp\}) \times \mathbb{B} \times T \times T$. 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 them will contain a valid timestamp from the set $T \setminus \{\perp\}$. We use $\mathcal{S}$ to denote the set of all possible credential state tuples. Throughout this paper, we will use dot notation to access fields of these state tuples (e.g., $s^e_c.r$ represents the receipt time of the credential whose state is stored in s^e_c).*

Definition 5 (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 each 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 of the global state and thus the consistency of an entity's view of the system becomes important to consider.

Definition 6 (Relevance). *A credential c is considered* relevant *to a policy P by entity e at time t if e has received c and considers the satisfaction of P in some way dependent on c at time t. Given a view V_e, $V_e^{P,t}$ is the subset of V_e containing state information for credentials that e considers relevant to P at time t.*

Definition 7 (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 6 is very subtle, as it can vary from user to user. For instance, a naive user might consider every credential that she has ever received to be relevant to a policy P, while another user might only consider credentials explicitly mentioned in P to be relevant. Further, the set of credentials considered relevant to a policy P by a single user might change over time. For example, if Alice is evaluating the policy $P = c_1 \wedge (c_2 \vee c_3)$, she 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 she may decide that c_1 and c_3 are relevant to P and again attempt to construct a consistent view. Consistency is fundamentally tied to the concept of relevance by Definition 7 and can thus be undermined by a faulty interpretation of relevance (for instance, by assuming that nothing is relevant to P). At a minimum, entities should consider the set of credentials used to satisfy P to be relevant to P and may also include other credentials in this set (for instance, credentials used to satisfy the release policies protecting credentials disclosed during the authorization protocol invoked to satisfy P). Resource providers have the autonomy and local knowledge necessary to decide which credentials are relevant at each moment and should thus be subjected to consistency requirements.

2.3 Practical Considerations for Consistency Enforcement

In this paper, we focus on limiting the unexpected behaviors that trust negotiation and distributed proving systems may 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.

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., seconds).

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.

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.

In the remainder of this paper, our discussion proceeds bearing these requirements in mind. In Section 5 we discuss how our solutions for enforcing view consistency satisfy these requirements.

3 Levels of Consistency

In this section, we present four increasingly-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 above reasons for credential invalidation can be expected to occur during the course of the authorization protocol. This indicates that like many other aspects of

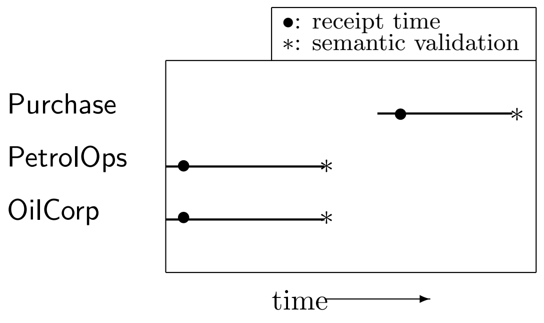

Figure 1: An incrementally consistent view.

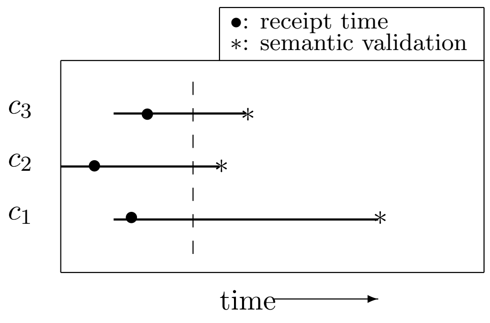

Figure 2: An internally consistent view.

trust negotiation and distributed proving, the choice of consistency level 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

To most people, the idea of satisfying a policy is straight forward. This usually entails presenting the evidence required to justify each clause of the policy. 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 $\mathit{checked} : \mathcal{S} \rightarrow \mathbb{B}$ and $\phi_{inc} : 2^{\mathcal{S}} \rightarrow \mathbb{B}$.

$$\mathit{checked}(s) \equiv (s.syn = \mathit{true}) \wedge (s.sem_v \neq \bot) \tag{1}$$
$$\phi_{inc}(V) \equiv \forall s \in V : \mathit{checked}(s) \wedge (\alpha(s.c) \leq s.r \leq s.sem_v) \tag{2}$$

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 1 is used when computing the endpoints of each credential's observed validity period. Thus, the formal definition of incremental consistency is as follows.

Definition 8 (Incremental Consistency). *A view $V_e^{P,t}$ is incrementally consistent iff $\phi_{inc}(V_e^{P,t})$ 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., [4, 5, 12, 23]) 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., [6, 14, 21, 24], to name a few).

Incremental consistency works especially well when authorization policies are stable predicates. A stable predicate is a condition that remains true once it becomes true. Example stable predicates include "Alice has paid her 2005 income taxes" and "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 1 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 inter-credential dependencies, such as that between Bob's PetrolOps and Finance credentials, incremental consistency is not always a good choice.

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 grid computing, 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 $start : 2^{\mathcal{S}} \rightarrow T$ and $end : 2^{\mathcal{S}} \rightarrow T$, and the predicate $\phi_{int} : 2^{\mathcal{S}} \rightarrow \mathbb{B}$.

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

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

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

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 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 9 (Internal Consistency). *A view $V_e^{P,t}$ is internally consistent iff $\phi_{int}(V_e^{P,t})$ 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

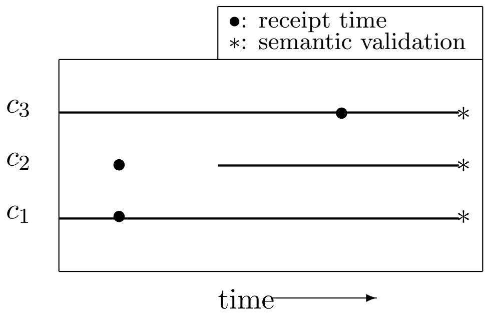

Figure 3: An endpoint consistent view.

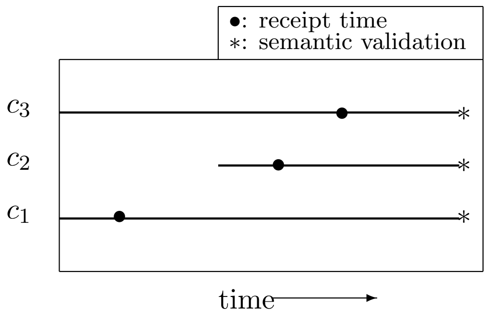

Figure 4: An interval consistent view.

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 2). 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 rather that they were all valid 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. We will call these levels of consistency *endpoint consistency* and *interval consistency*, respectively (see Figures 3 and 4). These consistency levels are defined in terms of the $\phi_{end} : 2^{\mathcal{S}} \rightarrow \mathbb{B}$ and $\phi_{interval} : 2^{\mathcal{S}} \rightarrow \mathbb{B}$ predicates.

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

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

Definition 10 (Endpoint Consistency). *A view $V_e^{P,t}$ is endpoint consistent iff $\phi_{end}(V_e^{P,t})$ is true at the decision point t.*

Definition 11 (Interval Consistency). *A view $V_e^{P,t}$ is interval consistent iff $\phi_{interval}(V_e^{P,t})$ is true at the decision point t.*

Guaranteeing the interval consistency of the view used to evaluate an authorization policy 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 inter-credential dependencies and external causes of revocation, that incrementally consistent and internally consistent views can actually be treated as endpoint consistent. Given the above definitions, it should be clear that the following proposition holds.

Proposition 3. *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 [18], the authors propose an access control 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 paper can 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" from these practical algorithms from the idealized case.

4.1 Comments on the Ideal Case

Each algorithm that we present in this paper, 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 $h : \{0,1\}^* \rightarrow \{0,1\}^l$ where l is the (fixed) output length of $h(\cdot)$.

Algorithm 1: Internal Consistency

```
1: // Initialize a connection with entity $e'$
2: Function Init($e' \in \mathcal{E}$) = Commit($e'$)
3:
4: // Commit credentials to entity $e'$
5: Function Commit($e' \in \mathcal{E}$) =
6: $s \leftarrow_r \{0,1\}^m$ // create a salt
7: $k \leftarrow k_e - |C_e|$ // need $k$ fake credentials
8: for $i = 1$ to $k$ do
9:     $r_i \leftarrow_r \{0,1\}^m$ // generate fake credentials
10: $CC_e \leftarrow \{h(s \mid c_1), \ldots, h(s \mid c_n), h(r_1), \ldots, h(r_k)\}$
11: Shuffle $CC_e$ randomly
12: Send $\langle e, s, CC_e \rangle$ to $e'$
13:
14: // Receive committed credentials from entity $e'$
15: Function Rcv($e' \in \mathcal{E}, s' \in \{0,1\}^m, CC_{e'} \in 2^{\{0,1\}^l}$) =
16: if $EntityInfo.lookup(e') \neq \perp$ then
17:     for all $\langle c, r, syn, sem_v, sem_i \rangle \in View$ do
18:         if $c$ is semantically valid then
19:             $View.store(c, \langle c, r, true, NOW, sem_i \rangle)$
20:         else
21:             $View.delete(c)$
22: $EntityInfo.store(e', \langle NOW, s', CC_{e'} \rangle)$
23:
24: // Receive a credential $c$ from entity $e'$
25: Function Rcv($e' \in \mathcal{E}, c \in \mathcal{C}$) =
26: $\langle rcv, s', CC_{e'} \rangle = EntityInfo.lookup(e')$
27: if $h(s' \mid c) \notin CC_{e'}$ then
28:     Reject $c$
29: else if (($c$ is syntactically valid) and ($\alpha(c) \leq rcv$) and
    ($c$ is semantically valid)) then
30:     $View.store(c, \langle c, NOW, true, NOW, \perp \rangle)$
31: else
32:     Reject $c$
```

Algorithm 2: Endpoint and Interval Consistency

```
1: // Receive a credential $c$ from entity $e'$
2: Function Rcv($e' \in \mathcal{E}, c \in \mathcal{C}$) =
3: if $c$ is syntactically valid then
4:     $View.store(c, \langle c, NOW, true, \perp, \perp \rangle)$
5: else
6:     Reject $c$
7:
8: // Invoked at the end of the access control protocol
9: Function ValidateAll($RelevantCreds \in 2^{\mathcal{C}}$) =
10: for all $\langle c, r, syn, sem_v, sem_i \rangle \in View$ do
11:     if $c \in RelevantCreds$ then
12:         if ($\omega(c) > NOW$) and ($c$ is semantically valid)
            then
13:             $View.store(c, \langle c, r, true, NOW, \perp \rangle)$
14:         else
15:             Fail and report that $c$ is invalid
```

Figure 5: Algorithms for view consistency enforcement.

- 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 $EntityInfo.store : \mathcal{E} \times (T \setminus \{\perp\}) \times \{0,1\}^m \times 2^{\{0,1\}^l} \rightarrow \perp$ stores state information. The function $EntityInfo.lookup : \mathcal{E} \rightarrow (T \setminus \{\perp\}) \times \{0,1\}^m \times 2^{\{0,1\}^l}$ retrieves state information.
- Each entity maintains a hash table, *View*, mapping credential identifiers to credential state information. The function $View.store : \mathcal{C} \times \mathcal{S} \rightarrow \perp$ stores credential state information, while $View.delete : \mathcal{C} \rightarrow \perp$ deletes state information.
- The current time as observed by some entity $e \in \mathcal{E}$ 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. 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 17–22).

This credential recommit process involves fairly high communication overheads for the recipient, as it must contact up to $|View|$ 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. *Any view created using Algorithm 1 is incrementally consistent.*

Proof. Lines 29–32 of Algorithm 1 ensure that each credential used during the execution of an authorization protocol is valid when it is received. This satisfies Definition 8 and thus any view created using Algorithm 1 is incrementally consistent. □

Proposition 5. *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) then 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 1. *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, 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 17–22 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 9 is satisfied and $V_e^{P,t}$ is internally consistent. □

Proposition 6. *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 adversary cannot tell how many newly acquired, but unused, credentials may be in a commitment set, so the third property holds. □

Although Theorem 1 asserts the soundness of Algorithm 1, this algorithm is not ideally complete as discussed 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 that 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 Appendix A, we propose an online credential status protocol that would allow 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 terminates the protocol if any credentials are found to be invalid. Because e has reached the 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 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 2. *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 12 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 1. It is therefore endpoint, internally, and incrementally consistent by Proposition 3. □

Although Algorithm 2 is sound (by Theorem 2) 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 [13, 9] lead us to conjecture that any sound and complete algorithm for endpoint consistency will require synchronized clocks.

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. 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 [6, 14]), 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 sub-negotiation 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 a 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 paper 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.

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 paper 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 paper 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 paper, 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 paper 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 the value could fluctuate between checks and not 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 paper 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 [3, 22, 25]. 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 levels of view consistency presented in this paper were designed to enforce various levels of safety, and thus the algorithms provided focused on satisfying only this criteria. However, safety may not always be the only concern for some resource providers. Rather, they may wish to enforce some level of safety but require algorithms with stronger guarantees regarding the availability of their services or privacy preservation than those provided by the algorithms in this paper.

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 [16] 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

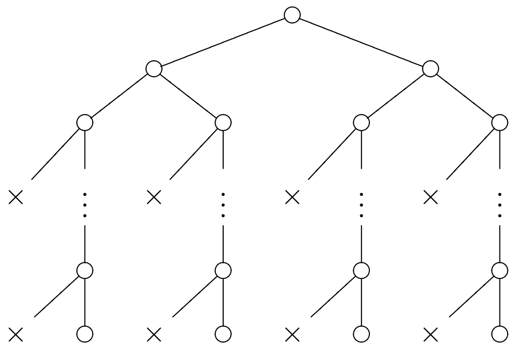

Figure 6: An example worst-case pruned binary tree for an instance of the Merkle commitment scheme in which an entity is committing four real credentials. Fake subtrees are denoted with a $\times$ character.

removed and replaced with a random string from $\{0,1\}^l$ 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 7. *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 6). 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 8. *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.*

Proof. As in the proof of Proposition 6, 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 simple induction, this prevents the remote party from

$\mathbf{k_e}$	**Number of actual credentials**					
	8	16	32	64	128	256
2^{16}	111 (0.3 ms)	207 (0.5 ms)	383 (0.6 ms)	703 (1.8 ms)	1279 (3.4 ms)	2303 (6.3 ms)
2^{32}	239 (0.7 ms)	463 (1.2 ms)	895 (2.8 ms)	1727 (4.7 ms)	3327 (8.9 ms)	6399 (16.8 ms)
2^{64}	495 (1.3 ms)	975 (2.8 ms)	1919 (5.2 ms)	3775 (10.3 ms)	7423 (19.8 ms)	14591 (38.9 ms)
2^{128}	1007 (2.7 ms)	1999 (5.4 ms)	3967 (10.5 ms)	7871 (20.9 ms)	15615 (45.0 ms)	30975 (83.0 ms)

Table 1: Required numbers of hashes and corresponding hash computation times for various configurations of the Merkle commitment algorithm.

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 1 contains the number of hash operations required generate Merkle commitments for between 8 and 128 actual credentials in sets containing between 2^{16} and 2^{128} potential credentials, along with the times required to compute these numbers of hashes.[1] These running times indicate that entities can easily commit their credentials into extremely large anonymity sets with only minimal computational and data transmission overheads when compared to those that would imposed by the commitment scheme used in Algorithm 1. When combined with Propositions 7 and 8, this shows that simply changing the commitment scheme used by Algorithm 1 allows us to tune both the performance and privacy guarantees of Algorithm 1 without effecting the consistency property that it enforces. This 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 A Note of Caution Regarding CA Clock Skew

The algorithms in this paper 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 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 (pre-issued 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 an amount of 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 widespread use of time synchronization protocols such as NTP [17] by service providers reduces the likelihood of this type of error randomly occurring between unrelated credentials. However, it is of the utmost importance that if two or more CAs issue mutually-exclusive certificates, their clocks be closely synchronized to ensure that no misleading apparent overlaps can occur. Such apparent overlaps are not introduced by the algorithms developed in this paper, 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 proactively altering at least one CA's clock prior to the issuance of some credential used in the negotiation that the attacker wishes to alter.

[1]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.

6 Related Work

Safety in trust negotiation has been discussed in several previous works, though the definitions of safety used in these works differs from that considered in this paper. In [25] Yu and Winslett describe 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 [22] 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 [11] 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 safety problems that emerge as a result of the consistency of the underlying state information used during policy evaluation rather than those that arise due to information leakage during a negotiation. 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 [20], [7], and [1], 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 transaction management in these domains typically involve the cooperation of multiple entities, as every entity has incentive to cooperate. However, as was mentioned 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 paper 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 is presented in [8]. This algorithm is not directly applicable to the problem addressed in this paper, 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 [2]), 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 paper, 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.

There are several 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 [25] 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.

Acknowledgments

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. Lee was also supported by a Motorola Center for Communications graduate fellowship. The authors wish to thank an anonymous reviewer who suggested the Merkle commitment scheme explored in Section 5.3.

References

[1] S. V. Adve and K. Gharachorloo. Shared memory consistency models: A tutorial. *IEEE Computer*, pages 66–76, Dec. 1996.

[2] O. Babaoğlu and K. Marzullo. Consistent global states of distributed systems: Fundamental concepts and mechanisms. In S. J. Mullender, editor, *Distributed Systems*, pages 55–96. Addison-Wesley, 1993. Also available as University of Bologna Technical Report UBLCS-93-1 at `http://www.cs.unibo.it/pub/TR/UBLCS/1993/93-01.ps.gz`.

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

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

[5] E. Bertino, E. Ferrari, and A. C. Squicciarini. Trust-X: A peer-to-peer framework for trust establishment. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):827–842, Jul. 2004.

[6] P. Bonatti and P. Samarati. Regulating service access and information release on the web. In *7th ACM Conference on Computer and Communications Security*, pages 134–143, 2000.

[7] W. Cellary, E. Gelenbe, and T. Morzy. *Concurrency Control in Distributed Database Systems.* Elsevier Science Publishing Company, Inc., 1988.

[8] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, Feb. 1985.

[9] D. R. Cheriton and D. Skeen. Understanding the limitations of causally and totally ordered communication. In *Proceedings of the ACM Symposium on Operating Systems Priniciples*, pages 44–57, 1993.

[10] R. Housely, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. IETF Request for Comments RFC-2459, Jan. 1999.

[11] K. Irwin and T. Yu. Preventing attribute information leakage in automated trust negotiation. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, Nov. 2005.

[12] H. Koshutanski and F. Massacci. Interactive credential negotiation for stateful business processes. In *3rd International Conference on Trust Management (iTrust)*, pages 257–273, May 2005.

[13] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, Jul. 1978.

[14] J. Li, N. Li, and W. H. Winsborough. Automated trust negotiation using cryptographic credentials. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*, pages 46–57, Nov. 2005.

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

[16] R. C. Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, Stanford University, 1979.

[17] D. L. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. IETF Request for Comments RFC-1305, Mar. 1992.

[18] K. Minami and D. Kotz. Scalability in a secure distributed proof system. In *Proceedings of the Fourth International Conference on Pervasive Computing (Pervasive 2006)*, May 2006.

[19] M. Myers, R. Ankney, A. Malpani, S. Glaperin, and C. Adams. X.509 Internet public key infrastructure online certificate status protocol - OCSP. IETF RFC 2560, Jun. 1999.

[20] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms.* Prentice Hall, Upper Saddle River, New Jersey, 2002.

[21] W. H. Winsborough and N. Li. Towards practical automated trust negotiation. In *Third IEEE International Workshop on Policies for Distributed Systems and Networks*, Jun. 2002.

[22] W. H. Winsborough and N. Li. Safety in automated trust negotiation. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2004.

[23] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu. The TrustBuilder architecture for trust negotiation. *IEEE Internet Computing*, 6(6):30–37, Nov./Dec. 2002.

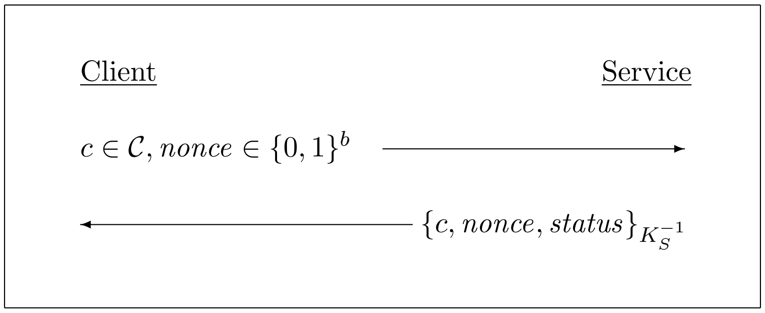
Client
Service
$c \in \mathcal{C}, nonce \in \{0,1\}^b$
$\{c, nonce, status\}_{K_S^{-1}}$

Figure 7: An online credential status protocol.

[24] M. Winslett, C. Zhang, and P. A. Bonatti. PeerAccess: A logic for distributed authorization. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS 2005)*, Nov. 2005.

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

[26] L. Zhou, F. B. Schneider, and R. van Renesse. COCA: A secure distributed online certification authority. *ACM Transactions on Computer Systems*, 20(4):329–368, Nov. 2002.

A Towards Completeness for Internal Consistency Algorithms

In this section, we propose an online credential status verification protocol that, when used in conjunction with Algorithm 1, would allow the modified Algorithm 1 to more closely approach ideal completeness. Figure 7 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 7. 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. Alice can now verify that the credential was valid at the time that it was committed by Bob.

Proposition 9. *If a credential c and its associated certified validity statement $cvs = \langle c, nonce, true\rangle$ 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.*

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. $\square$

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. This also 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 non-zero amount of time. As in that case, it is unlikely that ideal completeness could be reached without the assumption of synchronized clocks.