TOWARDS PRACTICAL AND SECURE DECENTRALIZED ATTRIBUTE-BASED AUTHORIZATION SYSTEMS

BY

ADAM J. LEE

B.S., Cornell University, 2003
M.S., University of Illinois at Urbana-Champaign, 2005

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2008

Urbana, Illinois

Doctoral Committee:

Professor Marianne Winslett, Chair
Professor Carl A. Gunter
Assistant Professor Nikita Borisov
Assistant Professor Cristina Nita-Rotaru, Purdue University

Abstract

The ubiquity of the Internet has led to increased resource sharing between large numbers of users in widely-disparate administrative domains. Unfortunately, traditional identity-based solutions to the authorization problem do not allow for the dynamic establishment of trust, and thus cannot be used to facilitate interactions between previously-unacquainted parties. Furthermore, the management of identity-based systems becomes burdensome as the number of users in the system increases. To address this gap between the needs of open computing systems and existing authorization infrastructures, researchers have begun to investigate novel attribute-based access control (ABAC) systems based on techniques such as trust negotiation and other forms of distributed proving.

To date, research in these areas has been largely theoretical and has produced many important foundational results. However, if these techniques are to be safely deployed in practice, the systems-level barriers hindering their adoption must be overcome. In this thesis, we show that safely and securely adopting decentralized ABAC approaches to authorization is not simply a matter of implementation and deployment, but requires careful consideration of both formal properties and practical issues. To this end, we investigate a progression of important questions regarding the safety analysis, deployment, implementation, and optimization of these types of systems.

We first show that existing ABAC theory does not properly account for the asynchronous nature of open systems, which allows attackers to subvert these systems by forcing decisions to be made using inconsistent system states. To address this, we develop provably-secure and lightweight consistency enforcement mechanisms suitable for use in trust negotiation and distributed proof systems. We next focus on deployment issues, and investigate how user interactions can be audited in the absence of concrete user identities. We develop the technique of virtual fingerprinting, which accomplishes this task without adversely affecting the scalability of audit systems. Lastly, we present TrustBuilder2, which is the first fully-configurable framework for trust negotiation. Within this framework, we examine availability problems associated with the trust negotiation process and develop a novel approach to policy compliance checking that leverages an efficient pattern-matching approach to outperform existing techniques by orders of magnitude.

To my parents.

Acknowledgments

I must first of all thank my advisor, Professor Marianne Winslett, for giving me the freedom to pursue topics of my own interest and providing me with *exactly* the amount of structure needed to ensure my success. You have not simply taught me how to succeed as a student, but rather how to be an independent researcher. Thank you so much for all of the academic, professional, and personal advice that you have given me, and for all of the hospitality that you and your family have shown me during the last five years.

I also want to express my gratitude to the members of my dissertation committee: Professors Carl A. Gunter, Nikita Borisov, and Cristina Nita-Rotaru. Your questions, comments, and advice have helped me grow as a researcher and have greatly improved the quality of my dissertation. Professors Roy H. Campbell, Klara Nahrstedt, and William H. Sanders provided a constant source of feedback and went out of their way to offer advice and assistance during the job search process. Dr. Jim Basney and Von Welch have also been valuable sources of information.

The enthusiastic support and mentoring that Dr. John Howard provided during my internships at Sandia National Laboratories sparked my interest in the research process and inspired me to continue my education. Dr. Kenneth J. Perano has been a valuable collaborator whose involvement in the development of TrustBuilder2 has greatly improved the quality of the system. Dr. Heidi Ammerlahn, Dr. Edward Talbot, and Jamie Van Randwyk provided valuable insight regarding potential applications of my research within the Department of Energy.

I thank my lab mates Sruthi Bandhakavi, Ragib Hasan, Soumyadeb Mitra, Rishi Sinha, Arash Termehchy, and Charles Zhang for sitting through numerous practice talks and for providing useful feedback on my work. Kazuhiro Minami has been a great source of ideas and inspiration. Jodie Boyer, Omid Fatemieh, Apu Kapadia, Fariba Khan, Michael LeMay, Prasad Naldurg, Geetanjali Sampemane, Parisa Tabriz and the other members of the Security Reading Group provided hours of fruitful discussion. I owe Jacob Biehl, Marina Blanton, Tanya Crenshaw, Shamsi Iqbal, and Erin Wolf-Chambers many, many thanks for carefully reading and commenting on numerous revisions of my application materials and provid-

ing so much support during the job search process.

Donna Coleman provided invaluable assistance in arranging travel, coordinating visitors, and navigating the maze of departmental regulations of which I was almost always blissfully unaware. I must also thank Anda Ohlsson for helping with the administrative duties associated with the Security Reading Group. Holly Bagwell, Angie Bingaman, and Mary Beth Kelley were helpful sources of advice about departmental procedures and provided many hours of entertaining conversation about the minutiae of graduate life.

The completion of this dissertation would not have been possible without the boundless encouragement and support of my family. My grandparents, Ann & Boris Kiryluk and Betty & Ned Lee, have always been there for me, even when it seemed that I would never get a "real job." There is no way to concisely thank you for a lifetime of love and support; I only wish that all of you were still here to share this moment with me. I also thank all of my aunts, uncles, cousins, great aunts, great uncles, in-laws, and close family friends for their kind words and encouragement over the years.

I am forever in debt to my brother, Chris, for being as solid as a rock throughout the difficult times that our family has faced while I was so many miles from home. You are a true friend.

My parents, Frederick and Christine Lee, have spent their lives encouraging my intellectual and personal growth. Many afternoons of listening to music or working in the yard with my mom have taught me to recognize the beauty that is always around us. Through countless hours of building and fixing all manner of things with my dad, I have learned to take pride in my work and to enjoy the simple pleasure of a job well done. Thank you both for everything that you have given me. Your love and generosity have shaped me into the person that I am today.

Last, but by no means least, I thank my wonderful wife, Stephanie Nelson, for bringing me so much laughter and happiness. Your patience, support, and affection throughout this process have helped me learn to seek a balance in life, rather than becoming mired in its inconsequential details. Thank you for the sacrifices that you have made, and for all of the love that you have shown me—our marriage is truly the greatest gift that I have ever received.

The work presented in this thesis was supported by a Motorola Center for Communications Graduate Fellowship; by a National Center for Supercomputing Applications Faculty Fellowship; by the National Science Foundation under grants IIS-0331707, CNS-0325951, and CNS-0524695; and by Sandia National Laboratories under grant number DOE SNL 541065.

Table of Contents

List of Tables

List of Figures

1 Introduction

On the Internet, nobody knows you're a dog.

—Peter Steiner

In theory, there is no difference between theory and practice.
In practice, there is.

—Yogi Berra

The secure operation of a distributed computing system requires an effective means of determining whether a given user is *authorized* to access the resources to which access is requested. Traditional solutions to the authorization problem for distributed systems have made implicit use of a *closed world* model, in which users and resources have a priori knowledge of one another. Under such an assumption, the problem of authorization reduces to that of *authentication*, since resource owners can simply maintain access control lists (ACLs) identifying the users that are authorized to access the resources that they manage. If a user Alice wishes to access some resource, she must first prove her identity to the resource owner, e.g., by using a username/password pair, X.509 identity certificate [61], or Kerberos ticket [99]. If Alice can be authenticated by the resource owner and she appears on the ACL for the resource, then she will be granted access to the resource.

An example identity-based authorization check is illustrated in Figure 1.1. In this example, Alice discloses her username and secret password to a resource provider at login time. The resource provider first checks to see that the password provided is actually the password assigned to Alice's user account. Since this check succeeds—and it is assumed that no one else knows Alice's password—a new session is started for the user that has authenticated as Alice. After successfully logging into the system, Alice issues a request to download the file `secret_note.txt`. The resource provider then checks the ACL for `secret_note.txt`. Since Alice appears on the ACL, a copy of the file is returned to her. This type of authorization system is elegant in its simplicity, and provides a very intuitive means of protecting resources in systems that fit this closed world model.

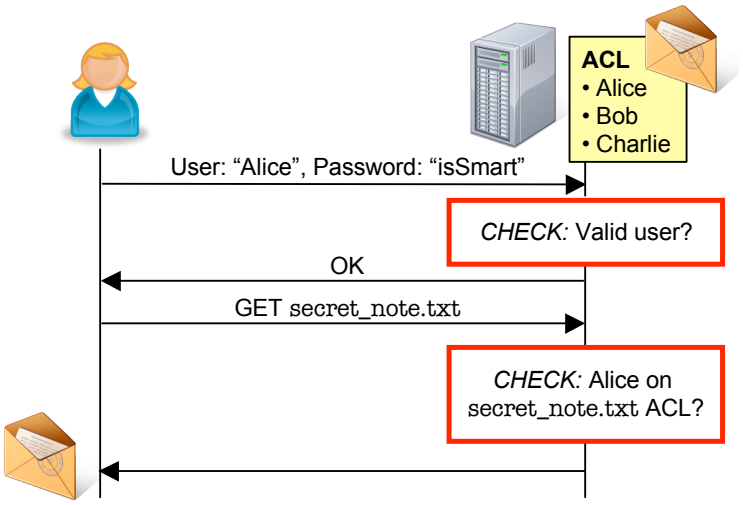

Figure 1.1: An access control list example.

While the closed-world assumption was reasonable to make in the early days of distributed computing, today's Internet is far from a closed world. Recent years have seen Internet technologies mature considerably, which provides a means for increasing numbers of traditionally "offline" services to establish themselves in the networked world. Furthermore, economic pressures, legislative requirements, and user demands for greater functionality and flexibility have provided impetus for the Internet to evolve from a network of closed systems into a more grassroots *open system*. The defining characteristics of such a system are that users have a need or desire to access resources outside of their own administrative domain, and resources have economic or other incentives to cater to these previously-unknown users. Examples of open systems include supply chain management networks, grid computing systems, pervasive computing environments, peer to peer networks, critical infrastructure and disaster management networks, electronic government services, and the diversity of "Web 2.0" applications that have become manifest in recent years. Although existing centralized and identity-based approaches to authorization have been used to support early incarnations of these types of services, this is a fundamentally flawed approach that cannot work in the long run.

The justification of the above statement is threefold. First, to a large extent, a user identity loses much of its significance outside of the domain in which it was issued. For example, a resource provider can probably determine that the user whose identity is `chris@uiuc.edu` is, in some way, affiliated with the University of Illinois. However, the resource provider cannot infer whether the user is male or female, let alone whether they are a student, faculty member, or staff member. Second, the trust established using legacy approaches is unilateral. That is, resources are

implicitly trusted and the burden of proof lies with the user to prove that he or she should have access to a given resource. In an open system, users may not have trust in previously-unknown resources that they discover, and thus there is a very real need for bilateral trust establishment during the authorization process. Lastly, identity-based approaches do not scale well in large open systems: managing lengthy ACLs is an error prone process and requiring users to safely manage a different username and/or password for each resource that they access is unrealistic. Even identity-based systems that allow the definition of user groups or roles to simplify access policy specification require centralized administration, which makes their use in decentralized open systems problematic.

To address the above limitations of identity-based access control approaches, researchers have investigated a number of decentralized attribute-based access control (ABAC) systems that retire the closed world assumption in favor of more realistic models of trust. Two popular decentralized ABAC approaches are *distributed proof* and *trust negotiation*, both of which will be described in detail in Chapter 2. Rather than relying on explicit access control lists, these types of authorization systems protect resources by using attribute-based access policies that describe the required *context* of the system and the *characteristics* of authorized users. These policies can be satisfied through the disclosure of cryptographic credentials issued by third-party attribute certifiers (e.g., professional organizations, employers, or government bodies) or other attestations made by devices within the system. Since both types of systems allow resource administrators to specify the *intension* of a policy, rather than its logical *extension* (i.e., an access control list), authorized entities can gain access to available resources without requiring that their identity be known a priori.

Distributed proof systems typically use a goal-oriented proof decomposition approach to find answers to authorization queries. For instance, in a pervasive computing space, a user might be allowed to turn on a digital projector if and only if a location service says that the user is located in the same room as the projector *and* a role service says that the user is allowed to assume the role "presenter." The media controller responsible for granting access to the projector can then collaborate with these other services to determine whether its access conditions are met. Trust negotiation systems allow the disclosure of sensitive attributes to be protected by release policies that place constraints on the resource owners to whom they can be disclosed. As such, a trust negotiation session evolves into a bilateral and iterative exchange of policies and credentials with the end goal of developing new trust relationships on-the-fly. This process is illustrated graphically in Figure 1.2 when Alice's reluctance to disclose her driver's license credential forces the negotiation

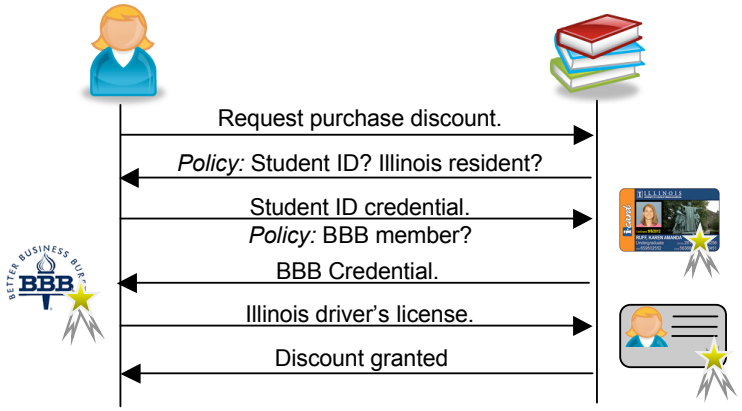

Figure 1.2: A trust negotiation session in which a user negotiates for a student discount at a bookstore.

into a second round, during which the bookstore must disclose its Better Business Bureau membership credential in an effort to "unlock" Alice's driver's license credential.

Recent research in decentralized ABAC systems has focused primarily on the theoretical aspects of these systems, producing foundational results on topics such as languages for expressing resource access policies (e.g., [17; 19; 57; 83]), strategies and tactics for constructing proofs of authorization (e.g., [12; 20; 68; 69; 90; 122]), logics for reasoning about the outcomes of distributed ABAC protocols (e.g., [24; 113; 119]), and formal treatments of the information leakages that can occur as a result of these types of protocols (e.g., [62; 116; 122]). Results such as these are absolutely essential to the development of satisfactory decentralized ABAC mechanisms. In fact, these and other results have established a very firm foundation upon which highly-expressive—yet rigorously understood—authorization systems can be built. However, this level of theoretical treatment does not address the myriad systems-level problems that can arise as these approaches are deployed in existing distributed computing environments.

In this thesis, we will show that safely and securely adopting decentralized ABAC approaches to authorization is not simply a matter of implementation and deployment, but requires careful consideration of both formal properties and practical issues. Our goal in this work is to provide sufficient evidence to justify the following thesis statement:

It is possible to develop extensible and efficient decentralized attribute-based access control systems that can be safely deployed in large-scale asynchronous distributed environments.

Successfully justifying the above claim will effectively bridge the gap between the theory and practice of decentralized ABAC approaches to authorization. To accomplish this goal, we will answer a progression of important questions related to the safety analysis, deployment, implementation, and optimization of these types of systems.

We begin by asking, *"Do existing theoretical models of trust negotiation and distributed proof protocols faithfully model asynchronous open systems, such as the Internet?"* Without such a guarantee, existing proofs regarding the soundness of these systems and possible avenues of information leakage may not hold in reality. In fact, we show that when the asynchronous nature of the Internet is combined with the autonomy granted to participants in these types of protocols, attackers *can* subvert existing decentralized ABAC systems in ways not captured by the existing theory. In particular, we show that an attacker can force a resource provider to sample inconsistent system states during protocol execution, thereby leading the resource provider to make unsafe decisions based upon a corrupted "view" of the system. The implication of this result is that existing decentralized ABAC protocols will, under certain circumstances, grant accesses to resources that would be denied by *any* centralized authorization mechanism. Although these problems are unsettling, we identify their root causes and develop lightweight concurrency control and consistency enforcement algorithms that eliminate this class of threats entirely.

After showing that existing decentralized ABAC protocols can be deployed in asynchronous environments without compromising their security properties, we turn our attention to more practical matters. Specifically, we ask, *"How can users be held accountable for their actions in a system without concrete user identities?"* Decentralized ABAC approaches represent a significant departure from the status quo, which implies that a transition from the centralized and identity-based method of securing and operating distributed systems may be problematic. Without a user identifier to bind to actions in the system, tasks such as auditing user behavior and establishing reputations become difficult. To address this problem, we propose a technique called *virtual fingerprinting*. A virtual fingerprint is an opaque pseudo-identifier derived from some subset of a user's digital credentials. Administrators can use virtual fingerprints to audit the actions of users in the system and can even share virtual fingerprints across security domains without leaking a user's sensitive attribute information. We show that virtual fingerprints can be used as a privacy-preserving basis for distributed audit systems, black lists, and reputation services in open systems lacking more traditional user identities.

The remainder of this thesis then focuses more directly on the specific case of trust negotiation systems. We ask, *"What are the key systems and architectural character-*

istics of the trust negotiation process, and what are their impacts on the performance of an organization's authorization infrastructure?" To date, implementations of trust negotiation systems have been largely unoptimized proofs of concept created to demonstrate the feasibility of specific protocol constructs. While they have performed admirably in this capacity, they were not designed for long-term use. As a result, these implementations do not interoperate with one another, nor do they provide a suitable framework for analyzing the performance characteristics of the trust negotiation process or attacks against these types of systems. To enable this type of analysis, we developed the TrustBuilder2 framework for trust negotiation. TrustBuilder2 is user-extensible by means of a dynamic type hierarchy and plug-in based architecture, meaning that support for new protocols, strategies, policy languages, and the like can be added with very little programmer overhead. Furthermore, it provides a framework within which the differences between various approaches can be quantitatively analyzed. Within this framework, we study the performance characteristics of the trust negotiation process, which leads us to identify its major bottlenecks and discover a new class of attacks against these systems.

The analysis that we conduct using TrustBuilder2 shows that checking compliance with ABAC policies is the most expensive portion of the trust negotiation process. This leads us to pose the question, *"Is it possible to increase the efficiency of the compliance checking process without altering its completeness or correctness properties?"* We show that the inefficiencies of this process stem from the fact that, for historical reasons, the policy compliance checking problem is often formulated as a theorem proving problem. While this is very logical, it carries with it undesirable overheads. To address these overheads, we show that it is possible to instead cast the policy compliance checking problem as a *pattern matching* problem. This enables the use of very efficient algorithms to determine *all* of the ways in which a user can satisfy a given policy. In addition to demonstrating that our approach is orders of magnitude faster than existing techniques, we formally prove that it provides *exactly* the same functionality as a theorem-proving approach to compliance checking, thereby ensuring the completeness and correctness of our improved process.

The research presented in this thesis makes a number of contributions to both the theory and practice of distributed authorization. In examining whether existing decentralized ABAC theory faithfully models asynchronous distributed systems, we make the following contributions:

- We present the first formalization of the view consistency problem for decentralized ABAC systems and show how the use of inconsistent views can lead to the permission of undesirable accesses.

- We formally characterize the causes of these types of problems and develop several levels of view consistency, each of which provides different guarantees regarding the types of undesirable accesses that can be prevented.
- We develop lightweight distributed algorithms that can be used to enforce each of our view consistency levels in practice. We then prove the soundness of these algorithms when used in asynchronous and adversarial environments.
- We demonstrate several other interesting properties of these algorithms including that they embody various privacy-preservation characteristics, have low computational and communication overheads, and can be used in conjunction with existing decentralized ABAC proposals.

While investigating the effects of decentralized ABAC approaches to authorization on user identifiability, we make the following contributions:

- We develop the notion of virtual fingerprints and show that they can be used in lieu of more traditional user identities as a basis for security support services such as audit logs, black lists, and reputation systems.
- We show that virtual fingerprints can be used in applications spanning multiple domains without leaking sensitive attribute information.
- To demonstrate the scalability of services based on virtual fingerprints, we develop Xiphos, a reputation system that is keyed on virtual fingerprints instead of traditional user identities. We carry out a simulation study to show that Xiphos performs well even in very large systems.

During our study of the systems and architectural properties of trust negotiation systems, we make the following contributions:

- We develop TrustBuilder2, which is the first fully-configurable framework for the design, deployment, and analysis of trust negotiation protocols. TrustBuilder2 demonstrates that a large number of trust negotiation proposals in the research literature can be unified under a single system architecture.
- We demonstrate that introducing a high degree of flexibility into advanced authorization frameworks does not necessarily incur high runtime overheads.
- Studies conducted using TrustBuilder2 lead us to identify the primary bottlenecks in the trust negotiation process and uncover a novel class of denial of service attacks that exploits the time required to check compliance with trust negotiation policies.

Lastly, we make the following contributions while studying ways to improve the performance of trust negotiation policy compliance checkers:

- We develop CLOUSEAU, a policy compliance checker that uses a pattern matching formalism to efficiently find *all* ways in which a policy can be satisfied. This approach outperforms existing theorem-proving approaches to compliance checking by orders of magnitude. As a concrete point of comparison, the compliance checker analyzed in [108] takes over 10 seconds to find two overlapping satisfying sets containing a total of 20 credentials, while CLOUSEAU finds the same satisfying sets in approximately 40 ms.
- CLOUSEAU improves not only the efficiency of the trust negotiation process, but also its utility. By determining all satisfying sets for a given policy, CLOUSEAU uncovers the entire "next-step" state space for any given negotiation. This allows negotiation strategies to make more intelligent decisions regarding how to proceed.
- We demonstrate that existing policy languages—such as RT_0, RT_1, and WS-SecurityPolicy—can be compiled into a format suitable for analysis by CLOUSEAU. This implies that policy writers can take advantage of the existing policy languages that best meet their needs without negatively influencing the runtime characteristics of trust negotiation implementations.

The rest of this thesis is organized as follows. Chapter 2 sets the stage for the remainder of the thesis by rigorously discussing trust negotiation and distributed proof construction approaches to authorization for open systems. Chapters 3 and 4 investigate the consistency and concurrency control issues that must be considered when deploying decentralized ABAC systems in asynchronous environments. Chapter 5 discusses how virtual fingerprinting can be used to enable the types of security support services that typically rely on more traditional notions of user identity. Chapter 6 presents the TrustBuilder2 framework for trust negotiation and investigates the systems and architectural characteristics of trust negotiation approaches to authorization. In Chapter 7, we explore a pattern matching approach to policy compliance checking that outperforms existing approaches by orders of magnitude. We then discuss related work in Chapter 8 and present our conclusions and directions for future work in Chapter 9.

2 Background

The ball is round, the game lasts 90 minutes, everything else is pure theory.

—Sepp Herberger

In this section, we set the stage for the rest of this thesis by describing two important decentralized ABAC approaches to authorization: *distributed proof construction* and *trust negotiation*. Section 2.1 focuses on distributed proof construction techniques. We first show how these types of systems evolved out of earlier trust management systems, and then examine the more advanced features supported by recent distributed proof systems. Section 2.2 then explains trust negotiation approaches to authorization, comparing and contrasting this approach with the distributed proof techniques discussed in Section 2.1.

2.1 Distributed Proof Construction

Modern distributed proof construction systems can be seen as direct descendants of earlier trust management systems. In 1996, Blaze, Feigenbaum, and Lacy coined the term *trust management* to refer to the study of the interplay between security policies, security credentials, and trust relationships [22]. In a trust management system—such as the PolicyMaker system described by Blaze et al.—credentials issued by principals in the system are converted into statements in some language of authorization. The policy derived from a collection of credentials is then analyzed in the context of a given service request to determine whether that request should be granted. A request is granted if and only if it is in compliance with the policy defined by the resource owner and the extensions to this policy entailed by the collection of credentials accompanying the request. As a result, the set of credentials accompanying a given request provides a formal proof of authorization in the event that the request is granted.

In some sense, trust management systems support a form of distributed proof, since the credentials provided as input to the system can be issued by any set of principals. However, early trust management systems—such as PolicyMaker [22;

23], KeyNote [21], SPKI/SDSI [49], and PCA [3; 15]—provided no mechanism for supporting the discovery of security credentials. Rather, the focus of these systems was more on policy representation and checking compliance with a policy *after* a set of credentials had been gathered. While this certainly provides a more distributed flavor of authorization than simple identity-based approaches, it is not the type of fully-distributed approach that is the subject of this thesis.

More recently, however, a number of distributed proof construction systems that support the runtime discovery of credentials have appeared in the research literature. For example, systems such as QCM [55], SD3 [64], Binder [44], Cassandra [17], the proof system developed by Minami and Kotz [90], and Grey [12] each allow security credentials to be collected at runtime and incorporated into the proof construction process. The specific mechanisms used by each of these proposals differ; however, this is largely immaterial to the discussion in this thesis. Rather than discussing the specifics of each of these systems, we will instead describe the salient features of the distributed proof construction process by means of an example protocol execution. When the details of a specific proof construction system or technique become relevant to this thesis, they will be described at that point.

2.1.1 An Example

In a distributed proof system, each principal maintains a local *knowledge base* that encapsulates its own view of the system. One portion of a principal's knowledge base is the set of *facts* that are currently known by the principal, which may include both *local* facts as well as digitally-signed *quoted* facts asserted by other principals. This set of facts is known as the principal's *extensional* knowledge base. A principal's knowledge base also contains a set of *derivation rules* that can be used to derive new facts from existing local facts and facts stored in the knowledge bases of other principals. The set of facts that can be derived using these derivation rules is known as the principal's *intensional* knowledge base. The decentralized nature of the distributed proof process makes it particularly well-suited for use in open systems, as the incomplete (and often complementary) views of many principals can be used to make decisions involving knowledge spread across multiple administrative domains.

To provide a better understanding of the distributed proof process, we will now examine the scenario depicted in Figure 2.1, which illustrates how the distributed proof process can be used to control access to resources in a pervasive computing environment. In this example, there are four principals whose knowledge bases are shown: a *media controller* that acts as a reference monitor protecting resources in

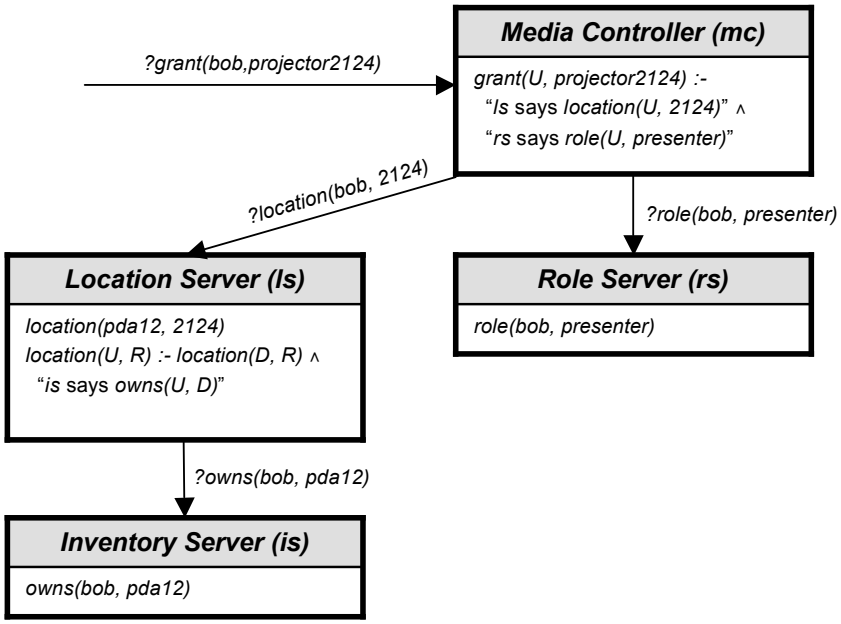

Figure 2.1: An example distributed proof system.

the system, a *location server* that monitors device locations, an *inventory server* that keeps track of bindings between devices and their human owners, and a *role server* that identifies the roles that can be assumed by specific users in the system. The knowledge bases of these principals need not necessarily contain both facts and rules.

Although the specific language used to express derivation rules varies from system to system, there are many similarities between these languages. For the purpose of the discussion in this section, derivation rules are expressed using a language similar to that used by the Binder proof system [44]. In the Binder system, derivation rules are written as Horn clauses that can refer to facts in a principal's local knowledge base and *quoted facts* in another principal's knowledge base. For instance, consider the following derivation rule taken from the knowledge base of the location server in Figure 2.1:

$$location(U, R) \text{ :- } location(D, R) \wedge \text{“}is \text{ says } owns(U, D)\text{”}$$

This rule allows the location service to conclude that a user U is located in room R if the location server knows that some device D is currently located in room R and the inventory server says that user U is the owner of device D.

In the scenario depicted in Figure 2.1, a user named Bob wishes to use a digital projector located in room 2124 of the pervasive computing space. To decide whether

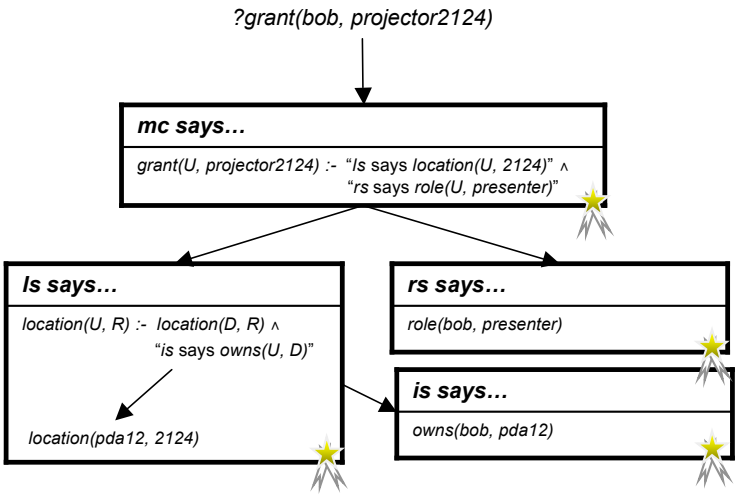

Figure 2.2: The proof tree generated during the example distributed proof.

to permit this request, the projector issues the query $?grant(bob, projector2124)$ to the media controller. This query matches the head of the media controller's rule protecting access to room 2124's projector via the substitution $\{U/bob\}$. As such, the media controller recursively issues two sub-queries to determine whether this derivation rule is satisfied: the query $?location(bob, 2124)$ is issued to the location server to determine whether Bob is currently located in room 2124 and the query $?role(bob, presenter)$ is issued to the role server to determine whether Bob is authorized to assume the "presenter" role in the pervasive computing space.

The query $?role(bob, presenter)$ is easily answered in the affirmative, since the fact $role(bob, presenter)$ is part of the role server's extensional knowledge base. Unfortunately, the query $?location(bob, 2124)$ does not match any fact in the location server's extensional knowledge base, as the location server tracks the location of *devices* in the system, not users. However, this query does match the location server's only derivation rule via the substitution $\{U/bob, R/2124\}$. As a result, the location server checks to see which devices are currently located in room 2124. The fact $location(pda12, 2124)$ reveals that the device $pda12$ is located in room 2124, so the location server recursively issues the query $?owns(bob, pda12)$ to the inventory server to determine whether Bob is the owner of $pda12$. This query succeeds because the fact $owns(bob, pda12)$ is in the inventory server's extensional knowledge base. As a result, the location server informs the media controller that Bob is indeed located in room 2124. When combined with the role server's earlier assertion regarding Bob's ability to assume the "presenter" role, this implies that Bob should be granted access to the projector in room 2124.

It is important to note that the format of the final proof returned to the querier

varies between distributed proof systems. In most systems, each principal participating in the proof construction process digitally signs the collection of rules and facts from its own knowledge base that were used to answer a given query, and returns them to the querier. In the event that the proof construction process involved recursive invocations of the protocol, these signed statements are passed up through each intermediate principal. The original querier can then verify that each digital signature is valid and that the collection of statements forms a valid proof tree whose leaves are facts and whose intermediate nodes are derivation rules. Figure 2.2 shows the proof tree that would be generated during the example proof execution discussed above.

Note, however, that recursively disclosing the above types of digitally-signed attestations requires the assumption that principals are willing to disclose the facts and rules in their knowledge bases to all other principals in the system. To address this problem, other proof systems—such as the Minami-Kotz system that we will discuss in Chapter 4—simply return the Boolean result of a given query. This allows portions of a proof tree to be hidden from the querier, provided that the querier trusts intermediate nodes to correctly execute sub-proofs on their behalf. In the future, other proof systems might allow for an intermediate approach in which principals could make a non-repudiable claim regarding a fact's status without disclosing the complete proof tree justifying their claim. The PeerAccess logical framework [119] can be used to reason about such a proof system, but to date, no such proof system has been implemented.

2.1.2 Research to Date

The majority of research concerning the use of distributed proof construction as an authorization approach has focused on languages for expressing access control policies or the design of proof construction tactics. However, several recent research efforts have begun to explore a number of other interesting facets of the distributed proof construction process. In [90], Minami and Kotz explore the design of an efficient distributed proof construction system (henceforth referred to as the MK system) for use in pervasive computing environments. One of the main differences between the MK system and other distributed proof systems is that the MK system permits principals to define access control lists (ACLs) protecting the disclosure of sensitive facts stored in their knowledge bases. This is important, as pervasive computing spaces have the ability to record large amounts of sensitive information regarding the users of the space. For instance, in the above example, the location server might opt to preserve users' location privacy by defining an

ACL permitting the disclosure of location data to the media controller, but not to any other principal in the system. In addition to this privacy preservation feature, the MK system supports the use of trusted intermediate caches with efficient revocation to speed up the proof construction and invalidation processes [91].

The MK system also allows the disclosure of encrypted facts. This allows proofs to be constructed even if certain participants in the proof process are not authorized to learn intermediate results. For example, consider the case in which the inventory server is willing to disclose facts of the form $owns(U, D)$ to the media controller, but not to the location service. In this case, the MK system would allow the inventory server to encrypt the value true using the public key of the media controller and then disclose this encrypted result to the location service. Since the truth value of the statement $location(bob, 2124)$ depends exclusively on this encrypted fact, the location server returns only this encrypted fact to the media controller. The media controller will ultimately decrypt the value true and determine that Bob is located in room 2124 *without* leaking the fact $owns(bob, pda12)$ to the location service. In [27], Borisov and Minami refine the use of encrypted facts to eliminate covert channel attacks against the MK proof system.

Researchers at Carnegie Mellon University have designed and deployed the Grey distributed proof system in an effort to study the interplay between distributed proof systems, users, and physical environments [11]. Within the Grey system, Bauer et al. have explored a number of interesting research topics including efficient tactics for constructing distributed proofs [12; 13], the usage of consumable credentials and one-time authorizations [28; 54], and using data mining techniques on access control logs to detect policy misconfigurations [14]. These researchers have also leveraged the user base of the Grey system to examine important issues surrounding the usability of distributed proof systems [10] and the facilitation of the policy creation process [9].

One issue that has not been considered by the above research efforts is that of temporal consistency. More specifically, existing proof systems do not ensure that the pieces of evidence collected to justify a given distributed proof are all true simultaneously. This is especially problematic in pervasive computing systems, since the context of a system—and thus the set of facts that are true at any given time—can change rapidly. The work presented in Chapters 3 and 4 of this thesis contributes to the distributed proof construction field by investigating efficient algorithms for ensuring the consistency of proofs generated during the distributed proof construction process. In Chapter 4, our techniques are applied to the MK proof system with minimal performance overheads.

2.2 Trust Negotiation

Trust negotiation [117] is an authorization approach in which resources are protected by attribute-based access policies, rather than explicit access control lists. For example, access to a particular computing cluster might be made available to graduate students studying computer science at an accredited university. Furthermore, each principal maintains some collection of digital credentials (e.g., X.509 certificates [61]) that attest to various attributes of the user or the context of his or her surrounding environment. These credentials are issued by third-party attribute certifiers, such as professional organizations, employers, or government bodies. For example, a typical graduate student might have a student ID credential issued by her university, a driver's license credential issued by the DMV, membership credentials issued by professional organizations such as the ACM and IEEE, and a host of project affiliation credentials issued by professors or funding organizations. This collection of credentials could then be used to grant her access to resources protected by ABAC policies, such as the computing cluster mentioned above.

At first glance, trust negotiation seems to be a simple distributed proof protocol in which all attributes relevant to a given user are managed by that user, rather than spread across the system. However, unlike in other distributed proof systems—such as those discussed in Section 2.1—a user's credentials are treated as first class resources that can be protected by *release policies* of their own. For example, the student discussed above may only wish to disclose her driver's license credential to principals who have a TRUSTe-certified privacy policy in place. Note that this is in contrast to most other distributed proof systems, which offer little or no protection for the facts used during the proof construction process. This flexibility to define attribute-based release policies for individual credentials gives users fine-grained, yet flexible, control over the disclosure of their potentially-sensitive attribute data. Furthermore, this allows for the incremental establishment of trust.

2.2.1 An Example

To better explain the details of the trust negotiation process, we will discuss the example trust negotiation session depicted in Figure 2.3. In this scenario, a student named Alice wishes to access a digital library portal on the World Wide Web. Upon requesting access to the digital library, Alice is returned a policy stating that she must be a graduate student at an accredited university and must be willing to disclose her permanent address (in the form of a driver's license or passport). Alice is willing to disclose her university affiliation to anyone, but for privacy reasons,

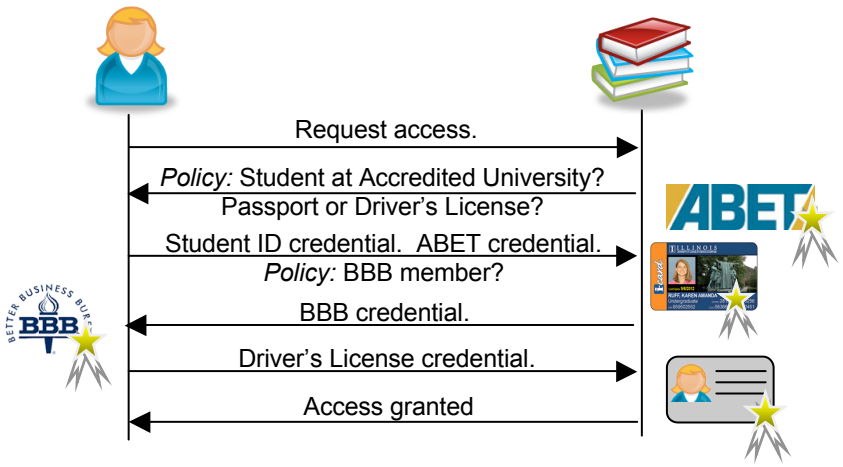

Figure 2.3: Using trust negotiation to access a digital library.

she is only willing to disclose her driver's license to members of the Better Business Bureau (BBB). As a result, Alice sends her student ID credential and a copy of her university's ABET certification to the digital library, along with a policy indicating that the digital library must be a BBB member. The digital library responds by returning a copy of its BBB membership credential, which satisfies Alice's policy. In response, Alice discloses a copy of her driver's license credential and is granted access to the digital library.

The trust negotiation process has a number of important benefits. First, it allows for the establishment of *bilateral* trust between the participants in the protocol. That is, either participant can request credentials from the other participant; this is in contrast to more traditional authorization systems in which the onus is on the user to prove that they are authorized to access the resource. Second, trust is established *incrementally*. In the above example, Alice released her attribute certificates to the digital library gradually, rather than all at once. Lastly, the trust negotiation process is *automated*. That is, policies can be verified and disclosures can be made by an agent process acting on a user's behalf.

A principal must not only disclose credentials during the trust negotiation process, but must also *prove ownership* of these credentials. For example, consider the case in which Alice's student ID credential is encoded as an X.509 certificate. In addition to disclosing this certificate to the digital library, Alice would digitally sign a challenge and disclose the result to the digital library. The digital library could use Alice's student ID credential to verify this challenge, which proves that the principal who disclosed the credential is in fact the owner of the credential. This proof of ownership mechanism prevents a malicious principal from attempting to pass off other principals' credentials as their own and thus plays an important role in the

trust negotiation process.

The above example trust negotiation session alludes to two important connections between trust negotiation and the types of distributed proof protocols discussed in Section 2.1. First and foremost, the end result of a trust negotiation session is a formal proof of authorization. Each resource is protected by a policy that identifies the attribute credentials that must be disclosed in order to access the resource, along with the trusted certifiers of those attributes. The collection of credentials that is eventually used to satisfy a policy is thus a formal proof of authorization, since the validity of these credentials can be checked by verifying that the issuer signature on each credential is correct. Second, the above example illustrates that new credentials may need to be obtained at runtime. Specifically, Alice is unlikely to have her university's ABET certification in her collection of local credentials. However, such credentials can be discovered at runtime and incorporated into the trust negotiation process by using certificate chain discovery techniques, such as those discussed in [55; 84].

2.2.2 Research to Date

Recent research in the trust negotiation area has been primarily of a theoretical nature, and has focused on a number of important issues that provide a strong formal foundation for the study of decentralized ABAC systems. A number of researchers have investigated the design of access control policy languages supporting a range of interesting features, including tunable constraint domains that can be used to adjust the expressiveness/efficiency trade-off [17], flexible role definitions and support for concepts such as separation of duty [83], the ability to represent both credentials and policies [19], and support for the credential chain discovery process [57; 84]. While the syntax of these languages can vary widely, most languages have a formal semantics based on either Datalog or constraint Datalog. As a result, the specific choice of policy language does not impact the facets of the trust negotiation process explored in this thesis.

Trust negotiation is an inherently strategy-driven process, since the choices made by participants in these protocols control the speed with which a negotiation takes place, as well as the amount of private information that is released. Therefore, the design and analysis of trust negotiation strategies and protocols is an area that has been pursued by a number of researchers with great success (e.g., see [20; 68; 69; 82; 115; 117; 122]). Analyzing the potential outcomes of the trust negotiation process is an important part of understanding the state of an open system. By understanding the initial states of users in the system, logics such as those presented in [24; 119]

allow administrators to reason with certainty about the outcome of a given access control protocol. The above results provide a strong theoretical foundation upon which *provably-secure* authorization systems can be designed, built, and verified.

Some of the techniques discussed in the trust negotiation literature have also been shown to be viable solutions for real-world systems through a series of implementations (such as those presented in [20; 58; 67; 118]) that demonstrate the feasibility of using these theoretical advances. However, after several years of research, trust negotiation protocols have yet to make their way into the mainstream. In the remainder of this thesis, we seek to facilitate the design, implementation, analysis, and optimization of trust negotiation systems. Specifically, we will address gaps that exist between the theoretical models of trust negotiation systems and real-life asynchronous distributed systems (Chapter 3), provide a means of auditing decentralized ABAC systems in the absence of concrete user identifiers (Chapter 5), develop a framework for investigating the systems issues associated with trust negotiation (Chapter 6), and show how the most expensive portion of the trust negotiation process can be greatly optimized without sacrificing correctness or completeness (Chapter 7).

3 Safety and Consistency in Certificate-Based ABAC Systems

It's not what you look at that matters, it's what you see.

—Henry David Thoreau

As discussed in Chapter 2, decentralized ABAC approaches are characterized by networked entities cooperating to form proofs of authorization 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 a variable-length window of time. Though the collections of credentials in some ways resemble partial system snapshots, current trust negotiation and distributed proof systems lack the notion of a consistent global state in which the satisfaction of authorization policies should be checked.

In this chapter, 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 providing increasingly-stringent 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 modifications to the core algorithms presented in this chapter. These modifications enhance the privacy-preservation or completeness properties of our algorithms without altering the consistency constraints that they enforce.[1]

[1]The material presented in this chapter was originally published as [73] and [77].

3.1 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 [17; 20; 69; 82; 83; 115; 118; 122] and distributed proof [12; 91; 119]. 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. In the remainder of this thesis we will refer to the collections of credentials gathered during decentralized ABAC protocols 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 [32]. 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 evaluation can be easily enforced. This relaxation of the semantics of policy satisfaction is especially worrisome in trust negotiation and distributed proof 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 3.1 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

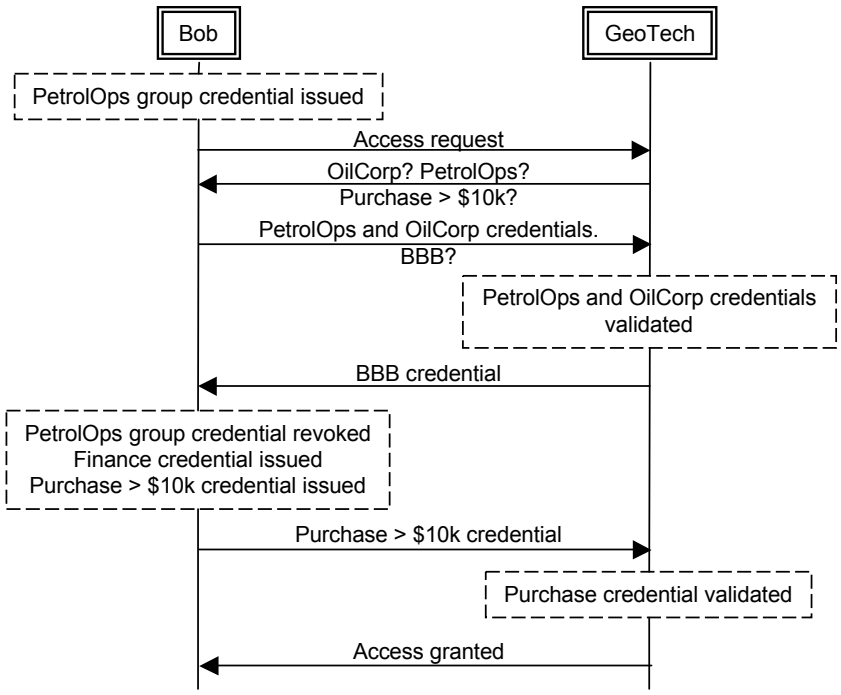

Figure 3.1: A graphical representation of Bob's interaction with GeoTech.

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.

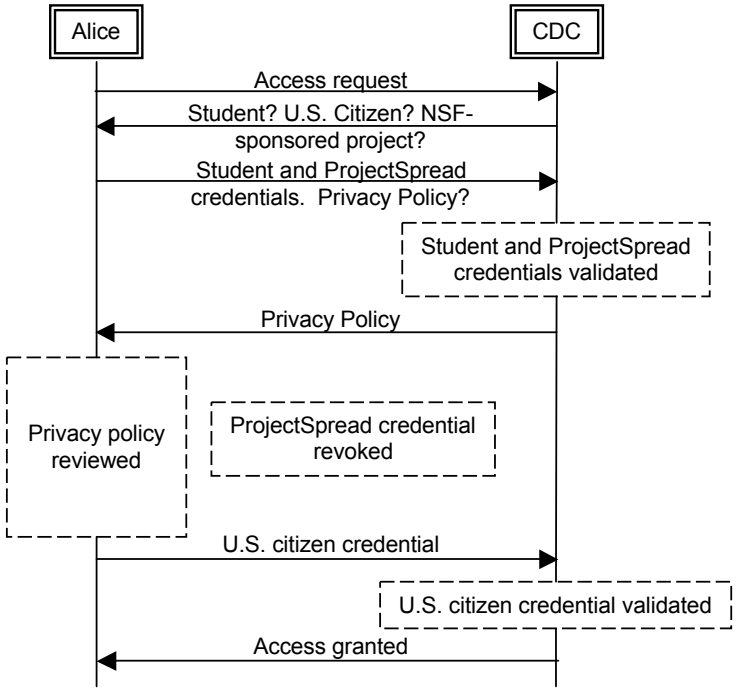

Figure 3.2: A graphical representation of Alice's interaction with the CDC.

Example 2 Figure 3.2 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 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 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. Since access control policies encode safety properties that must be preserved by the system, we refer to the above types of problems as *safety problems*. The rest of this chapter focuses on developing efficient and provably correct methods for verifying that *consistent* views of the system are used during policy evaluation (i.e., views encoding combinations of credential states that actually existed), thereby ensuring that no unsafe accesses are granted.

To the best of our knowledge, the problem of enforcing view consistency in trust negotiation and distributed proof systems has not been discussed elsewhere in the literature. Though similar to the consistency problems studied in distributed systems [110], distributed databases [31], 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 chapter 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 should 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 chapter, 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 proof. We present the first formalization of the view consistency problem for trust negotiation and distributed proof 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 3.2). 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 3.3). We provide algorithms that can be incorporated into existing trust negotiation and distributed proof systems to attain these levels of consistency and prove the correctness of each algorithm (Section 3.4). 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 proof protocol and no cooperation between groups of CAs or other users (Section 3.5). Finally, we summarize this contributions of this chapter (Section 3.6).

3.2 System Assumptions and Problem Definition

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

3.2.1 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 proof will be used for this purpose, as the lack of pre-existing trust relationships in the system prevents the use of traditional identity-based authorization mechanisms.

We place no limitations on the temporal duration of a trust negotiation or distributed proof 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 [82; 122]; 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 [12; 91; 119]) 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 3.1). In the remainder of this chapter, 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 [61], $\alpha(c)$ 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 all inter-credential dependency constraints existing between credentials that it has issued are respected 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 (i.e., whether the credential is valid or revoked). This functionality could be provided through the Online Certificate Status Protocol (OCSP) [94] or by an online CA such as COCA [123].

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

DEFINITION 3.2.1 (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 3.2.2 (Semantic Validity). *A credential c 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 $\alpha(c) \leq t \leq t'$.*

Informally, if a credential is syntactically valid, then it is well-formed. The semantic validity of a credential at a given time means 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 statements encoded in the credential are 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 for a credential that contains 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 chapter.

DEFINITION 3.2.3 (Validity). *A credential c is* valid *at time t if it is syntactically and semantically valid at time t.*

PROPOSITION 3.2.4. *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 3.2.5. *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 3.2.6. *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 observed earlier, each credential collected by an entity during a trust negotiation or distributed proof 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 proof session.

DEFINITION 3.2.7 (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 $s_c^e = \langle c, r, syn, sem_v, sem_i \rangle \in \mathcal{C} \times (T \setminus \{\perp\}) \times \mathbb{B} \times T \times T$. 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 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 thesis, we will use dot notation to access fields of these state tuples. For example, $s_c^e.r$ represents the receipt time of the credential whose state is stored in the tuple s_c^e.

DEFINITION 3.2.8 (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 of the global state and thus the consistency of an entity's view of the system becomes important to consider.

DEFINITION 3.2.9 (Relevance). *A credential c is considered* relevant *to a policy P by entity e at time t if e has received c and 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 to be relevant to P at time t.*

DEFINITION 3.2.10 (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 3.2.9 is very subtle, as the concept of relevance 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 3.2.10 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 cre-

dentials 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.

3.2.3 Practical Considerations for Consistency Enforcement

In this chapter, we focus on limiting the unexpected behaviors that trust negotiation and distributed proof 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.

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 proof 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 proof protocols.

We will bear these requirements in mind throughout the remainder of this chapter; in Section 3.5 we will discuss the ways in which our solutions for enforcing view consistency satisfy these requirements.

3.3 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 3.1 can be expected to apply during the course of the authorization protocol. This indicates that like many other aspects of trust negotiation and distributed proof, 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 3.5.

3.3.1 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 $\mathit{checked} : \mathcal{S} \rightarrow \mathbb{B}$ and $\phi_{inc} : 2^{\mathcal{S}} \rightarrow \mathbb{B}$.

$$\mathit{checked}(s) \equiv (s.syn = \mathit{true}) \wedge (s.sem_v \neq \bot) \tag{3.1}$$

$$\phi_{inc}(V) \equiv \forall s \in V : \mathit{checked}(s) \wedge (\alpha(s.c) \leq s.r \leq s.sem_v) \tag{3.2}$$

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

DEFINITION 3.3.1 (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

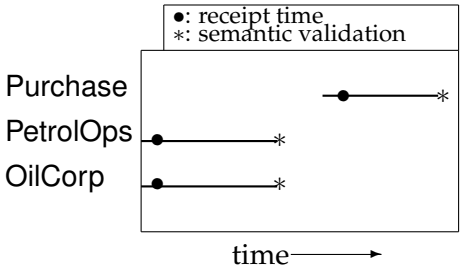

Figure 3.3: An incrementally consistent view.

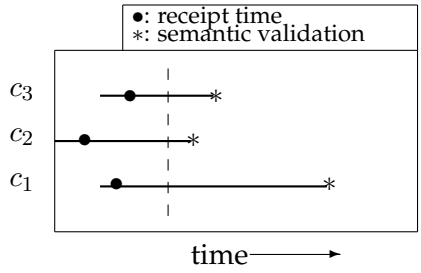

Figure 3.4: An internally consistent view.

(e.g., [17; 20; 69; 118]) 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 proof that, to the best of our knowledge, have not yet been implemented (e.g., [24; 82; 115; 119], 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 3.1. Figure 3.3 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 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 proof 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.

3.3.2 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 $\phi_{int} : 2^{\mathcal{S}} \rightarrow \mathbb{B}$.

$$start(V) = min(\{s.r \mid s \in V\}) \tag{3.3}$$

$$end(V) = max(\{s.r \mid s \in V\}) \tag{3.4}$$

$$\begin{aligned} \phi_{int}(V) \quad \equiv \quad & (\forall s \in V : checked(s)) \\ & \wedge (max(\{\alpha(s) \mid s \in V\}) < min(\{s.sem_i \mid s \in V\})) \\ & \wedge (max(\{\alpha(s) \mid s \in V\}) < end(V)) \\ & \wedge (min(\{\omega(s) \mid s \in V\}) > start(V)) \end{aligned} \tag{3.5}$$

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 3.3.2 (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

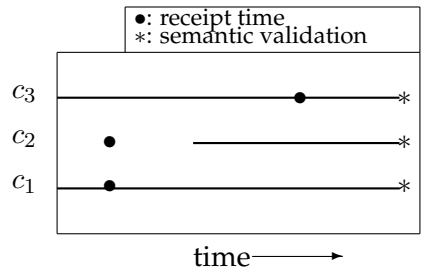

Figure 3.5: An endpoint consistent view.

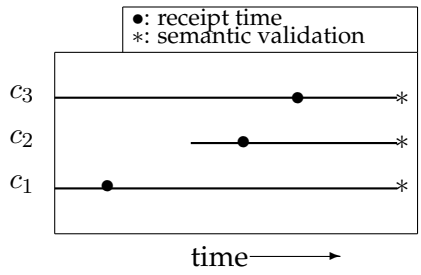

Figure 3.6: An interval consistent view.

each credential's validity interval (see Figure 3.4). 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.

3.3.3 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 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 3.5 and 3.6). 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.

$$\phi_{end}(V) \equiv \forall s \in V : checked(s) \wedge (\alpha(s.c) \leq end(V) \leq s.sem_v) \tag{3.6}$$

$$\phi_{interval}(V) \equiv \forall s \in V : checked(s) \wedge (\alpha(s.c) \leq s.r \leq end(V) \leq s.sem_v) \tag{3.7}$$

DEFINITION 3.3.3 (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 3.3.4 (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 3.3.1 and 3.3.2, we showed that if certain assumptions could be made about the likelihood of inter-credential 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 3.3.5. *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 [91], 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 3.2.1, 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 chapter can be proven sound without making such assumptions.

3.4 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.

3.4.1 Comments on the Ideal Case

Each algorithm that we present in this chapter, 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.

3.4.2 Internal Consistency

Algorithm 3.1 ensures that the views used for authorization policy evaluation are internally consistent. We make the following assumptions in Algorithm 3.1 (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 $h(\cdot)$.
- Each entity e chooses a parameter k_e used to hide the number of credentials that she possesses.
- Each entity maintains a hash table, $\mathit{EntityInfo}$, mapping entity names to state information. The function $\mathit{EntityInfo.store} : \mathcal{E} \times (T \setminus \{\perp\}) \times \{0,1\}^m \times 2^{\{0,1\}^l} \rightarrow \perp$ stores state information. The function $\mathit{EntityInfo.lookup} : \mathcal{E} \rightarrow (T \setminus \{\perp\}) \times \{0,1\}^m \times 2^{\{0,1\}^l}$ retrieves state information.
- Each entity maintains a hash table, View, mapping credential identifiers to credential state information. The function $\mathit{View.store} : \mathcal{C} \times \mathcal{S} \rightarrow \perp$ stores

Algorithm 3.1 Internal Consistency

```
1: // Initialize a connection with entity e'
2: Function INIT(e' ∈ E) = COMMIT(e')
3:
4: // Commit credentials to entity e'
5: Function COMMIT(e' ∈ E) =
6: s ←_r {0,1}^m // create a salt
7: k ← k_e − |C_e| // need k fake credentials
8: for i = 1 to k do
9:     r_i ←_r {0,1}^m // generate fake credentials
10: CC_e ← {h(s | c_1), ..., h(s | c_n), h(r_1), ..., h(r_k)}
11: Shuffle CC_e randomly
12: Send ⟨e, s, CC_e⟩ to e'
13:
14: // Receive committed credentials from entity e'
15: Function RCV(e' ∈ E, s' ∈ {0,1}^m, CC_e' ∈ 2^{{0,1}^l}) =
16: if EntityInfo.lookup(e') ≠ ⊥ then
17:     for all ⟨c, r, syn, sem_v, sem_i⟩ ∈ View do
18:         t ← NOW
19:         if c is semantically valid then
20:             View.store(c, ⟨c, r, true, t, sem_i⟩)
21:         else
22:             View.delete(c)
23: EntityInfo.store(e', ⟨NOW, s', CC_e'⟩)
24:
25: // Receive a credential c from entity e'
26: Function RCV(e' ∈ E, c ∈ C) =
27: t ← NOW
28: ⟨rcv, s', CC_e'⟩ = EntityInfo.lookup(e')
29: if h(s' | c) ∉ CC_e' then
30:     Reject c
31: else if ((c is syntactically valid) and (α(c) ≤ rcv) and (c is semantically valid)) then
32:     View.store(c, ⟨c, t, true, t, ⊥⟩)
33: else
34:     Reject c
```

credential state information, while $View.delete : \mathcal{C} \rightarrow \perp$ deletes state information.

- The current local time is accessible via the local variable NOW.
- Each entity detects the failure of other participants in the algorithm by using timeouts. If the initiator of an algorithm detects such a failure, the algorithm can be aborted and restarted. To simplify the presentation of our algorithms, further details of this process are omitted.

Algorithm 3.1 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 17–23).

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 3.1.

PROPOSITION 3.4.1. *Any view created using Algorithm 3.1 is incrementally consistent.*

Proof. Lines 31–34 of Algorithm 3.1 ensure that each credential used during the execution of an authorization protocol is valid when it is received. This satisfies Definition 3.3.1 and thus any view created using Algorithm 3.1 is incrementally consistent. □

PROPOSITION 3.4.2. *All credentials accepted by Algorithm 3.1 were held by their bearer at the time of the most recent credential recommit.*

Proof. For Algorithm 3.1 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 3.1 were held by their bearer at the time of the most recent credential recommit. □

THEOREM 3.4.3. *If e's execution of a trust negotiation or distributed proof protocol for target policy P succeeds at time t while using Algorithm 3.1 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 3.4.1, 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 proof sessions requiring up to $n-1$ invocations of the COMMIT method. If the trust negotiation or distributed proof 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 3.3.2 is satisfied and $V_e^{P,t}$ is internally consistent. □

PROPOSITION 3.4.4. *Algorithm 3.1 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 exact 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 3.4.3 asserts the soundness of Algorithm 3.1, this algorithm is not ideally complete as defined in Section 3.4.1. That is, it is possible for all cre-

Algorithm 3.2 Endpoint and Interval Consistency

```
1: // Receive a credential c from entity e'
2: Function RCV(e' ∈ ℰ, c ∈ 𝒞) =
3: if c is syntactically valid then
4:     View.store(c, ⟨c, NOW, true, ⊥, ⊥⟩)
5: else
6:     Reject c
7:
8: // Invoked at the end of the access control protocol
9: Function VALIDATEALL(RelevantCreds ∈ 2^𝒞) =
10: for all ⟨c, r, syn, sem_v, sem_i⟩ ∈ View do
11:     if c ∈ RelevantCreds then
12:         t ← NOW
13:         if (ω(c) > NOW) and (c is semantically valid) then
14:             View.store(c, ⟨c, r, true, t, ⊥⟩)
15:         else
16:             Fail and report that c is invalid
```

dentials to be valid simultaneously at the time of the last recommit even if Algorithm 3.1 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 3.5.5, we propose an online credential status protocol that allows Algorithm 3.1 to more closely approximate ideal completeness.

3.4.3 Endpoint and Interval Consistency

Algorithm 3.2 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 [31]. 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 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 3.4.5. *If an execution of a trust negotiation or distributed proof protocol for a target policy P succeeds at time t while using Algorithm 3.2 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 3.2.6. It is therefore endpoint, internally, and incrementally consistent, by Proposition 3.3.5. □

Although Algorithm 3.2 is sound (by Theorem 3.4.5) 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 3.2. The well-known limitations of causal orderings and virtual clocks [36; 70] lead us to the following assertion regarding the ideal completeness of endpoint and interval consistency algorithms.

THEOREM 3.4.6. *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 3.4 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 3.7 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 credential was observed valid

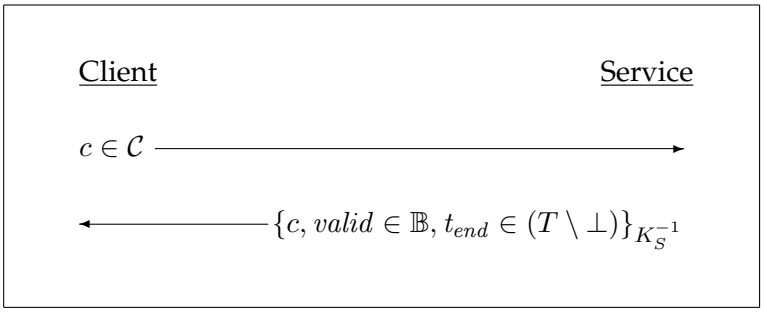

Figure 3.7: An online credential status protocol leveraging synchronized clocks.

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.

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 3.7 is used to make the semantic validity check that occurs on line 13 of Algorithm 3.2, 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 3.2 relying on a certificate validation protocol like the Online Certificate Status Protocol (OCSP) [94]. 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 3.2 using the protocol presented in Figure 3.7 is still sound (by Theorem 3.4.5), this shows that clock synchronization is a sufficient condition for constructing a sound and ideally complete endpoint or interval consistency enforcement algorithm. □

3.4.4 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 3.2 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 [24; 82]), 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 3.1 can be modified to call the VALIDATEALL method at its conclusion. However, Algorithm 3.1 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 3.1), 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.

3.5 Discussion

In this section, we discuss several interesting facets of view consistency. In particular, we show that the algorithms presented in this chapter satisfy the requirements presented in Section 3.2.3, consider the effects of an unstable environment on view consistency, and introduce the notion of strategic algorithms for view consistency enforcement. We then comment on the effects of poorly synchronized CA clocks and propose a novel online credential status protocol that allows Algorithm 3.1 to very closely approach ideal completeness.

3.5.1 Requirements Revisited

In Section 3.2.3 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 chapter 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 3.1, 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 3.2 requires no cooperation between entities in the system to succeed. Lastly, the algorithms presented in this chapter have virtually no impact on existing trust negotiation and distributed proof 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 chapter, existing systems can enforce stronger levels of view consistency.

3.5.2 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 its 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 chapter 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 chapter 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.

3.5.3 Strategic Algorithm Design

Trust negotiation and distributed proof are dynamic processes, the properties of which depend on the strategies or tactics adopted by their participants [12; 116; 122]. 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 3.4.4). The levels of view consistency presented in this chapter were designed to enforce various levels of safety, and thus our algorithms focused on satisfying only this criterion. 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 (i.e., they require an algorithm that closely approaches ideal completeness) or privacy preservation than those provided by the algorithms in this chapter.

For instance, recall that Algorithm 3.1 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 [88] 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 3.1.

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 $\{0,1\}^l$, henceforth referred to as a *fake subtree*; recall from Section 3.4.2 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 3.5.1. *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.*

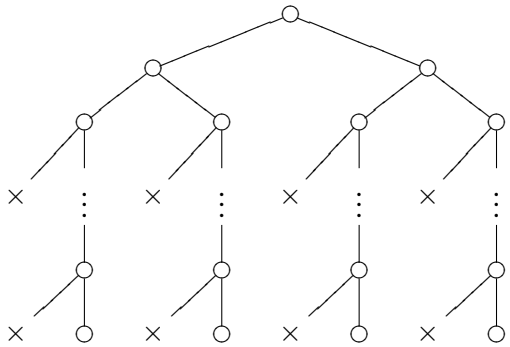

Figure 3.8: An example worst-case pruned binary tree for an instance of the Merkle commitment scheme in which an entity is committing four real credentials. Fake subtrees are denoted by the $\times$ symbol.

Proof. 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 3.8). 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 3.5.2. *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 exact 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 3.4.4, the preimage resistance property of $h(\cdot)$

$\mathbf{k_e}$	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)

Table 3.1: Required numbers of hashes and corresponding hash computation times for various configurations of the Merkle commitment algorithm.

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 3.1 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 3.1. When combined with Propositions 3.5.1 and 3.5.2, this shows that changing only the commitment scheme used by Algorithm 3.1 allows us to tune both the performance and privacy guarantees of the algorithm without affecting 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.

3.5.4 A Note of Caution Regarding CA Clock Skew

The algorithms in this chapter 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 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 [89] 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 chapter, 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.

3.5.5 Towards Completeness for Internal Consistency Algorithms

We now propose an online credential status verification protocol that, when used in conjunction with Algorithm 3.1, allows the modified Algorithm 3.1 to more closely approach ideal completeness. Figure 3.9 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 3.1'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 3.9. 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 autho-

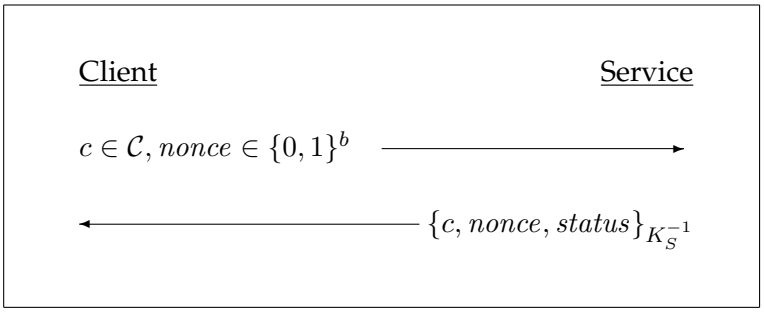

Figure 3.9: An online credential status protocol leveraging causal orderings.

rization protocol, he must also disclose its associated certified validity statement to Alice. Figure 3.10 illustrates how this protocol extends the commitment phase of Algorithm 3.1 to become a four-step process. Note that although only the initial commitment of Algorithm 3.1 is shown in Figure 3.10, 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 3.5.3. *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.*

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 3.2.5). □

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 3.1 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 3.1 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

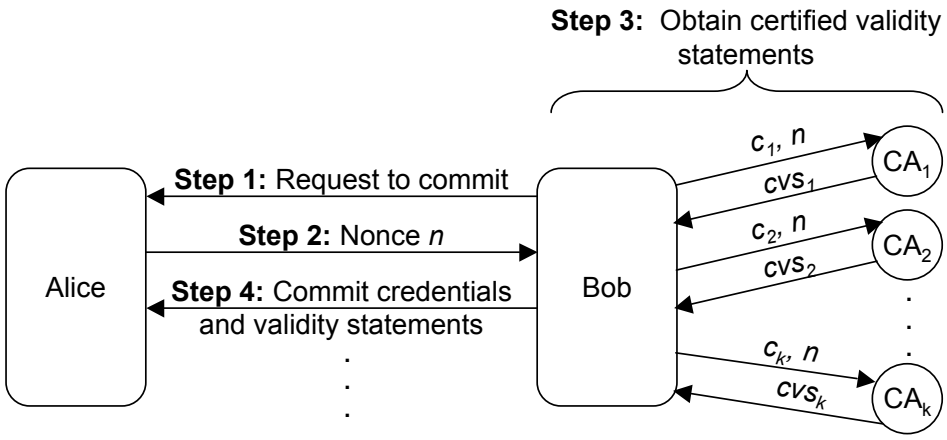

Figure 3.10: An illustration of how the protocol presented in Figure 3.9 can be integrated with the commitment phase of Algorithm 3.1.

to the problem discussed in Section 3.4.3 in which Algorithm 3.2 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.

3.6 Summary

In this chapter, we presented the notion of view consistency in decentralized ABAC 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 3.4 that enhance the privacy-preservation properties of these algorithms or their closeness to ideal completeness without altering the consistency constraints that they enforce.

4 Generalized Safety and Consistency Models

It is one of the commonest of mistakes to consider that the limit of our power of perception is also the limit of all there is to perceive.

—C.W. Leadbeater

In distributed proof construction systems, information release policies can make it unlikely that any single node in the system is aware of the complete structure of any particular proof tree. This property makes it difficult for queriers to determine whether the proofs constructed using these protocols sampled a consistent snapshot of the system state; in Chapter 3, we showed that this can have undesirable consequences in decentralized authorization systems. Unfortunately, the consistency enforcement solutions that we developed in Chapter 3 were designed for systems in which only information encoded in certificates issued by certificate authorities is used during the decision-making process. Further, we made the assumption that each piece of certified evidence used during proof construction is available to the decision-making node at runtime.

In this chapter, we generalize our previous results and present lightweight mechanisms through which consistency constraints can be enforced in distributed proof systems in which the full details of a proof may be unavailable to the querier and the existence of certificate authorities for certifying evidence is unlikely; these types of distributed proof systems are likely candidates for use in pervasive computing and sensor network environments. We present modifications to one such distributed proof system that enable three types of consistency constraints to be enforced while still respecting the same confidentiality and integrity policies as the original proof system. Further, we detail a performance analysis that illustrates the modest overheads of our consistency enforcement schemes.[1]

[1]The material presented in this chapter was originally published as [71] and [72].

4.1 Introduction

The process of making informed authorization decisions in dynamic environments where trust relationships cannot be determined a priori is widely accepted as a difficult task. This is particularly true in context-rich environments such as pervasive computing spaces, as the set of permissible actions may depend on the physical context of the space. This context can be sampled through the use of sensors deployed throughout the environment. To address this complexity, several rule-based systems have been designed for specifying and checking authorization policies in pervasive computing environments (e.g., see Al-Muhtadi et al. [2], Bacon et al. [5], Covington et al. [41], and Myles et al. [95]). Recently, frameworks for constructing and validating *distributed* proofs have been proposed to address the limitations of using centralized knowledge bases for making authorization decisions [12; 90; 119].

In authorization systems based on distributed proving, resource access requests are permitted if a resource owner can construct a well-formed proof tree whose root is a logical statement granting the requester access to the resource. The topology of a proof tree shows the logical dependencies among the facts in the tree; that is, the leaves of this tree represent base facts, while intermediate nodes represent inferences made using these facts. Such a proof tree need not be formed solely from facts in the resource owner's local knowledge base; subtrees of a proof may be produced by other entities in the network provided that the resource owner trusts the integrity of information provided by these entities (e.g., see Bauer et al. [12], Minami and Kotz [90], and Winslett et al. [119]). In some systems, information release policies may prevent portions of a subproof from being revealed to certain nodes in the proof tree [90]. An important observation is that the logical leaves of a distributed proof tree form one possible *view* of the state of the environment in which the proof was constructed. Resource access is granted because, in that view of the system, it was possible to construct a proof tree justifying the access request. If the facts making up a proof tree represent stable assertions (i.e., facts whose validity will not change), then this view is actually a snapshot of the system and the semantics of policy satisfaction remain the same as in centralized proof systems. However, if *any* facts in the proof tree are not constant, then in some circumstances, it is possible to form a proof tree justifying access to a particular resource that would have been denied in *any* centralized system. That is, an inconsistent view can lead a prover to think that certain logical facts were true simultaneously when, in fact, they were not. Clearly, this can lead to the permission of undesirable accesses to system resources.

For example, consider a hospital wired with sensors such as occupancy detectors,

location tracking devices, and door lock sensors. Now, a clinician, Alice, decides to use the projector located in her office to review the medical records of several patients that she is working with. In order for the system to permit the use of the projector to view medical records, it must be the case that the occupancy of Alice's office is one, Alice is located in her office, and the door to her office is locked. When Alice requests this access, the system might first check that the occupancy of her office is one and then proceed to check that Alice is currently located in her office. As this check is being made, Bob enters the room and closes the door behind him, which automatically locks. The system determines that Alice is located in her office and then checks that the door is locked; since the door is locked the medical records are displayed on the projector. This is a clear violation of the policy protecting patient records that might have legal ramifications, as Bob may not be authorized to view the records being projected. In addition to this type of accidental violation of system view consistency, intentional attacks on the system are also possible.

The adverse effects of inconsistent views on authorization systems were previously examined in Chapter 3. In that chapter, we focused on studying the properties of systems in which all attestations used during proof construction were encoded in certificates issued by one or more trusted Certificate Authorities (CAs). The solutions for enforcing the use of consistent states presented in Chapter 3 relied on the timing and sequencing of checks for certificate revocation that can be made using protocols such as OCSP [94] or COCA [123]. Unfortunately, the effectiveness of these solutions hinges directly on this supporting infrastructure for checking credential validity. As a result, these solutions cannot be used in proof construction frameworks that rely on simple digital signatures or keyed MACs to authenticate proof facts, including many of those designed to be used in pervasive computing or sensor network environments.

In this chapter, we build upon these previous consistency enforcement results and show how to ensure that distributed proofs constructed using these more general forms of trusted information can be formed by sampling consistent system states without impeding on the autonomy of nodes in the system (e.g., by requiring participation in a wide-scale transaction-management protocol). Further, we present solutions to the consistency problem that work even if some details of a proof tree are hidden from the query issuer by information flow policies; recall that the solutions presented in Chapter 3 assumed that the policy evaluator had complete knowledge of the proof tree formed during the protocol. Although we focus our presentation on authorization systems based on distributed proving, the techniques described in this chapter are applicable to any system in which autonomous entities wish to leverage decentralized information to make decisions in a poten-

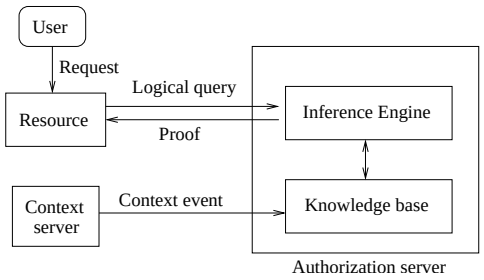

Figure 4.1: Structure of an authorization server.

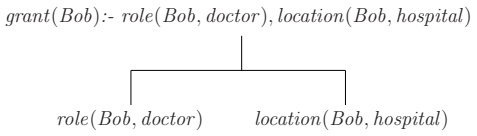

Figure 4.2: An example distributed proof tree.

tially adversarial environment.

The rest of this chapter is organized as follows. In Section 4.2, we overview background material regarding the distributed proof construction protocol that we will modify to enforce view consistency constraints. Section 4.3 formally defines our system model and the levels of view consistency that we wish to enforce in this Chapter. In Section 4.4, we present modifications to an existing distributed proof construction protocol to enable the use of three types of consistent views when making authorization decisions. Further, we present proofs that the security and privacy properties of the underlying proof system have not been altered by our modifications. We quantitatively evaluate the performance impact of our consistency enforcement schemes in Section 4.5, and then summarize this chapter in Section 4.6.

4.2 Background

In this section, we discuss the distributed proof construction protocol presented by Minami and Kotz [90], as later sections of this chapter focus on modifying this protocol to ensure that authorization decisions are made using consistent states. Rather than presenting this proof system in its entirety, we discuss several examples that illustrate the key features of this system; interested readers can refer to [90] for a more in-depth treatment of this proof construction system. We chose to explore the consistency problem within the context of this protocol as it allows portions of a proof tree to be hidden from certain entities participating in the construction of the proof tree, including the node issuing the query, whereas most other distributed proof frameworks assume that the querying node gathers all supporting evidence locally prior to making a decision. The techniques developed in this chapter for use in the Minami-Kotz proof construction system can also be applied to other distributed proof systems with less restrictive properties.

4.2.1 Structure of the Authorization Server

Figure 4.1 shows the structure of an authorization server consisting of a knowledge base and an inference engine. The knowledge base stores both authorization policies and facts including context information. The context server publishes context events and updates facts in the knowledge base dynamically. The inference engine receives authorization queries from remote servers, such as resource servers processing users' access requests. The inference engine then attempts to derive logical proofs justifying these queries using the facts in its local knowledge base and possibly even interactions with remote parties. If the inference engine cannot construct a proof, it returns a proof that contains a false value. In the open environment of pervasive computing, each server could belong to a different administrative domain.

Rules and facts in a knowledge base are represented as a set of Horn clauses in Prolog. For example, a medical database may define an authorization policy that requires a requester P to hold a role membership "doctor" and to be physically located at the "hospital" as follows.

$$grant(P) \text{ :- } role(P, doctor), location(P, hospital)$$

The atoms $role(P, doctor)$ and $location(P, hospital)$ on the right side of the clause are the conditions that must be satisfied to derive the granting decision $grant(P)$ on the left. If a user *Bob* issues a request to read a medical database, the proof tree in Figure 4.2 could be constructed based on the above rule. The root node in the tree represents the rule and the two leaf nodes represent the facts. Notice that the variable P in the rule is replaced with a constant *Bob*. A user's location, which is expressed with the $location$ predicate, is a dynamic fact; i.e., the second variable of the predicate $location$ should be updated dynamically as Bob changes his location.

4.2.2 Proof Decomposition

Multiple authorization servers in different administrative domains can cooperate to handle authorization queries in a peer-to-peer manner. These peer-to-peer interactions are guided by each entity's *integrity policies*, which specify sets of entities trusted to handle particular types of queries. For example, if Alice specifies the integrity policy $trust(location(P, L)) = \{Bob\}$, then she trusts Bob to accurately answer queries regarding the location of other entities. In the most basic case, the principal who issues a query trusts the principal who handles this query in terms

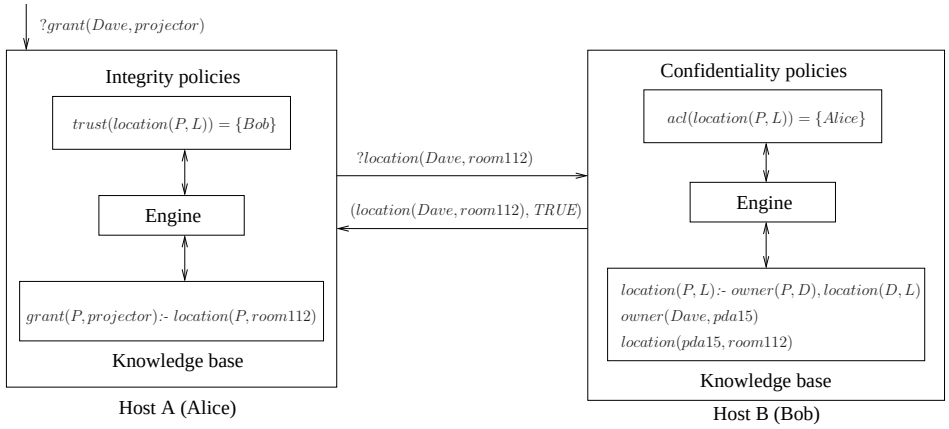

Figure 4.3: Remote query between two principals. Alice is a principal who maintains a projector, and Bob is a principal who runs a location server.

of the integrity of the query result. As such, the handler principal need not disclose the entire proof tree that she generates; she needs only to return a proof that states whether the fact in the query was true. In general, however, the querier may not *completely* trust the query handler and thus her integrity policies might place constraints on the rules used by the handler to generate the proof tree. In this case, a more complete proof tree, whose intermediate nodes are digitally signed, would need to be returned by the handler. This way, the querier can verify that her integrity policies were respected.

Figure 4.3 describes one possible collaboration between a querier and handler. Suppose that host A run by principal Alice, who owns a projector, receives an authorization query $?grant(Dave, projector)$ that asks whether Dave is granted access to that projector. Since Alice's authorization policy in her knowledge base refers to a requester's location (i.e., $location(P, room112)$), Alice issues a query $?location(Dave, room112)$ to host B run by Bob. Alice chooses Bob, because Bob satisfies Alice's integrity policies for queries of the type $location(P, L)$. Bob processes the query from Alice, because Alice satisfies Bob's confidentiality policies for queries of the type $location(P, L)$ as defined in Bob's policy $acl(location(P, L)) = \{Alice\}$. Bob derives the fact that Dave is in *room112* from the location of his device using the facts $location(pda15, room112)$ and $owner(Bob, pda15)$. However, he only needs to return a proof that contains a single root node that states that $location(Dave, room112)$ is true, because Alice believes Bob's statement about people's locations (i.e., $location(P, L)$), according to her integrity policies. The proof of the query is thus decomposed into two subproofs maintained by Alice and Bob.

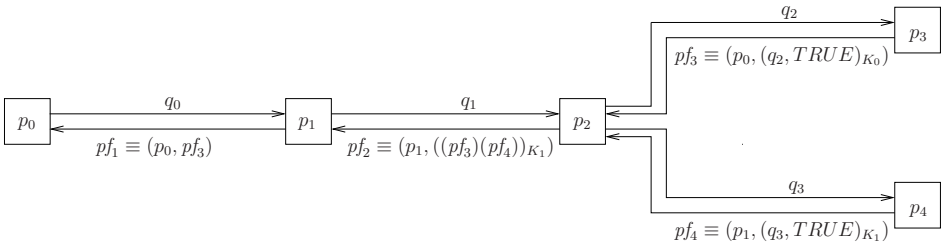

Figure 4.4: Enforcement of confidentiality policies. The first item in a proof tuple is a receiver principal, and the second item is a proof tree encrypted with the receiver's public key.

4.2.3 Enforcement of Confidentiality Policies

Each fact provider maintains a set of *confidentiality policies* that determine which entities are authorized to receive the facts that she provides. These policies are enforced by encrypting a query result (along with a querier-provided nonce to ensure freshness) using the public key of an authorized receiver. Each query is accompanied by a list of upstream principals who could possibly receive the answer of the query; this enables the handler to choose an authorized recipient from the list of upstream principals that satisfies her confidentiality policies. It is therefore possible to obtain an answer for some initial query even when some number of intermediate principals in the distributed proof do not satisfy the confidentiality policies of a fact provider. Figure 4.4 shows an example collaboration among principals p_0, p_1, p_2, and p_3. When principal p_0 issues an authorization query q_0 to principal p_1, p_1 issues a subsequent query q_1, which causes principal p_2's queries q_2 and q_3. Since a receiver principal of a proof might not be the principal who issued the query, a reply for a query is a tuple $(p_i, (pf)_{K_i})$ where p_i is an identity of a receiver principal and $(pf)_{K_i}$ is a proof encrypted with the receiver's public key. We associate a receiver principal identity with an encrypted proof so that a principal who receives an encrypted fact can decide whether to attempt to decrypt that encrypted fact. We assume that, in this example, each principal who issues a query trusts the integrity of the principal who receives that query in terms of the correctness of whether the fact in the query is true or not. For example, p_0's integrity policies contain a policy $trust(q_0) = \{p_1\}$.

Suppose that query q_1's result (i.e., true or false) depends on the results of queries q_2 and q_3, which are handled by principals p_3 and p_4, respectively, and that p_3 and p_4 choose principals p_0 and p_1, respectively, as receivers since p_2 does not satisfy

their confidentiality policies. Because principal p_2 cannot decrypt the results from principals p_3 and p_4, p_2 encrypts those results with the public key of principal p_1, which p_2 chose as a receiver. Principal p_2 forwards the encrypted results from p_3 and p_4 because the query result of q_1 is the conjunction of those results. Principal p_1 decrypts the encrypted result from p_2 and obtains the encrypted results originally sent from principals p_3 and p_4. Since p_1 is a receiver of the proof from p_4, p_1 decrypts the proof that contains a true value. Since a query result for q_0 depends on the encrypted proof from p_3, principal p_1 forwards it in the same way. The principal p_0 finally decrypts it and obtains an answer for query q_0. The key observation here is that principal p_0 is not aware of the fact that the query result is originally produced by principal p_3.

This proof system applies public-key operations only to a randomly generated symmetric key and uses that symmetric key to encrypt and decrypt a proof; that is, a proof consists of a new symmetric key encrypted with a receiver's public key and a proof encrypted with that symmetric key. In addition to the public-key encryption, the querier and handler principals use another shared symmetric key to protect other data fields (e.g., a receiver identity) in a query and a proof from eavesdroppers. We assume that the two principals share the symmetric key via a protocol using public-key operations when the querier and handler principal authenticate with each other for the first time.

4.3 Definitions

We begin this section by describing the system model within which the Minami-Kotz distributed proof construction protocol discussed in Section 4.2 was designed to be used. We then show that existing solutions to the view consistency problem are not applicable due to fundamental differences between system models. Lastly, we formally define the view consistency problem within the context of the system model presented in this section and then define four important view consistency levels.

4.3.1 System Model

Distributed proof construction protocols were designed to be used in open-system environments consisting of a possibly infinite set of autonomous entities, $\mathcal{E}$. Each entity $e \in \mathcal{E}$ possesses one or more public key certificates that can be used to authenticate messages signed by e or to encrypt messages that are to be sent to

e. These certificates are made publicly available by one or more key servers or through the use of decentralized peer-to-peer protocols. Without loss of generality, we will assume that each node uses only one public key certificate during the construction of any single distributed proof. We place no limitations on the temporal duration of executions of the proof construction protocol, nor do we assume any level of clock synchronization between entities in $\mathcal{E}$.

All evidence used during the construction of a distributed proof takes the form of assertions signed with the providing entity's private key. As described in Section 4.2, the proof construction process is assumed to be policy-directed. Each entity maintains a collection of *integrity policies* that indicate which other entities are trusted to answer different types of queries; adherence to these integrity policies can be checked by verifying the signatures on responses to any issued subqueries. Each entity e also maintains a collection of *confidentiality policies* that control the release of subproofs generated by e. This interplay between integrity policies and confidentiality policies implies that the complete details of a proof tree may not be available to entities in the system. In particular, the querying entity will not learn any details of the proof tree beyond those specified by his or her integrity policies. Further, intermediate nodes in a proof tree may not learn whether the proof construction protocol was successful, as the results of subqueries issued by these nodes may be hidden from them by the query target's confidentiality policies (see Section 4.2.3 for an example of this behavior). These assumptions imply that the view consistency solutions developed in Chapter 3 cannot be used in this environment.

4.3.2 Problem Definition

As observed in Section 4.1, the use of inconsistent views of a system during policy evaluation can lead to situations in which a policy evaluator believes that certain facts were true simultaneously when, in fact, they were not. We now more precisely define this problem.

DEFINITION 4.3.1 (Validity). *An entity e can determine that some proof fact f is* valid *at time t if either (i) f is in e's local knowledge base at time t, or (ii) f is considered valid at time t by a remote entity who is trusted to provide information regarding f.*

Asserting the validity of some fact f at a particular time t by invoking case (ii) of the above definition is not a straightforward task. Consider the case where some entity e issues a query for fact f to another entity e' at time t_{iss}. Due to delays in the network and processing delays at e and e', it is likely that e' will not receive

the query until some time $t' > t_{iss}$. Similarly, e is unlikely to receive e''s response to his query until some time $t_{rcv} > t'$. Therefore, e cannot conclude that f was valid at either t_{iss} or t_{rcv}; he can only infer that f was valid at *some* time t where $t_{iss} \leq t \leq t_{rcv}$. We will discuss methods for fine-tuning these types of inferences later in this chapter. As facts are collected and validated, an entity builds a *view* of the system that will be used to construct a proof of authorization.

DEFINITION 4.3.2 (Fuzzy Validity Interval). *The interval $[t_s, t_e]$ is a* fuzzy validity interval *for some fact f if f can be shown to be valid at some (possibly unknown) time t such that $t_s \leq t \leq t_e$.*

DEFINITION 4.3.3 (Concrete Validity Interval). *The interval $[t_s, t_e]$ is a* concrete validity interval *for some fact f if f can be shown to be valid at all times t such that $t_s \leq t \leq t_e$.*

DEFINITION 4.3.4 (Fact State). *Let the set T contain all possible time stamps, let $\perp$ denote the null value, and let ℓ be a predefined length parameter. The* fact state *for a fact f as observed by some entity is denoted by the five-tuple $s = \langle id, e, t_\alpha, t_\omega, fuzzy \rangle \in \{0,1\}^\ell \times \mathcal{E} \times T \cup \{\perp\} \times T \cup \{\perp\} \times \mathbb{B}$. The value id is an ℓ-bit identifier assigned to the fact f (which may simply be an encoding of that fact), e identifies the entity from which f was obtained, t_α and t_ω are local timestamps, and $fuzzy$ is a Boolean value indicating whether $[t_\alpha, t_\omega]$ specifies a fuzzy ($fuzzy =$ **true**) or concrete ($fuzzy =$ **false**) validity interval. The set of all possible fact state tuples is denoted by $\mathcal{S}$.*

Entities in the system create fact state tuples as the validity of certain facts is revealed during the execution of the distributed proof protocol. In the remainder of this chapter, we will use dot notation to access the fields of fact state tuples. For instance, $s.id$ represents the identifier of the fact whose state is stored in s. If a fact state tuple s for a fact f has either of its $s.t_\alpha$ or $s.t_\omega$ fields set to $\perp$, then no conclusions can be drawn about the validity status of f.

DEFINITION 4.3.5 (View). *A* view *is a finite set of fact state tuples containing no more than one tuple for any $\langle id, e \rangle$ pair.*

An entity e's view of the system is a set of local observations that e has made regarding the validity of certain facts. Given that any such view contains only *local* observations, it is unlikely to capture a precise snapshot of the entire system state. As such, the consistency level of a view is of central importance. Although a view may contain data associated with any number of facts, unless noted otherwise, we assume without loss of generality that an entity e will only wish to enforce consistency constraints on views comprised of facts associated with a single distributed proof.

DEFINITION 4.3.6 (View Consistency). *A view V is said to be ϕ-consistent if and only if V satisfies some predicate ϕ that places temporal constraints on the observed validity intervals of the facts whose state data are stored in V.*

4.3.3 Levels of Consistency

We now describe three increasingly-stringent levels of view consistency relevant to distributed proof construction protocols for use in the system model described in Section 4.3.1. We then discuss a fourth view consistency level designed specifically for use in pervasive computing environments.

Incremental Consistency

The most basic definition of view consistency that one can imagine is what we will refer to as incremental consistency. Intuitively, an incrementally consistent view is a view in which each fact was valid at some point during the construction of the related proof tree. To formally define the notion of incremental consistency, we first define the predicates $checked : \mathcal{S} \rightarrow \mathbb{B}$, $fuzzy : \mathcal{S} \rightarrow \mathbb{B}$, and $concrete : \mathcal{S} \rightarrow \mathbb{B}$.

$$checked(s) \equiv (s.t_\alpha \neq \perp) \wedge (s.t_\omega \neq \perp) \wedge (s.t_\alpha \leq s.t_\omega) \tag{4.1}$$

$$fuzzy(s) \equiv checked(s) \wedge s.fuzzy \tag{4.2}$$

$$concrete(s) \equiv checked(s) \wedge \neg s.fuzzy \tag{4.3}$$

The predicate $checked(s)$ ensures that the fact state tuple s contains a fully-defined validity interval. The $fuzzy(s)$ predicate is true if and only if s encodes a fully-defined fuzzy validity interval; $concrete(s)$ is true if and only if s encodes a fully-defined concrete validity interval. Given these predicates, we can now formally define the notion of incremental consistency for distributed proof systems via the predicate $\phi_{inc} : 2^{\mathcal{S}} \times T \times T \rightarrow \mathbb{B}$ as follows:

$$
\begin{aligned}
\phi_{inc}(V, t_s, t_e) \quad \equiv \quad & \forall s \in V : checked(s) \\
& \wedge (fuzzy(s) \rightarrow \\
& \quad ((t_s \leq s.t_\alpha) \wedge (s.t_\omega \leq t_e))) \\
& \wedge (concrete(s) \rightarrow \\
& \quad ((s.t_\alpha \leq t_s \leq s.t_\omega) \vee \\
& \quad (s.t_\alpha \leq t_e \leq t_\omega))
\end{aligned} \tag{4.4}
$$

The predicate ϕ_{inc}—which is a reformulation of the predicate ϕ_{inc} defined in Chapter 3—is satisfied by a view V during some interval $[t_s, t_e]$ if and only if each fact state tuple in the view contains a fully-specified validity interval, each fuzzy validity interval is a subinterval of $[t_s, t_e]$, and each concrete validity interval overlaps $[t_s, t_e]$ at some point. This gives us the following definition for an incrementally consistent proof construction, and an associated theorem.

DEFINITION 4.3.7 (Incremental Consistency). *A view V generated between the time that a given query was issued, t_{iss}, and the time that the completed proof tree was received by the issuer, t_{rcv}, is* incrementally consistent *if and only if $\phi_{inc}(V, t_{iss}, t_{rcv})$ is true.*

THEOREM 4.3.8. *The Minami-Kotz distributed proof construction protocol always uses incrementally consistent views when evaluating authorization policies.*

Proof. Assume that the distributed proof construction algorithm succeeds in constructing a proof tree using a view that is not incrementally consistent. This implies that there exists some fact f that was not true at any point during execution of the proof construction protocol. This means that the validity status for f (which must be true for the proof to succeed) was contributed to the proof tree *before* the proof construction process was started or, equivalently, was a replayed validity status from an earlier execution of the protocol. However, each validity status returned by a fact provider is causally-linked to the query executed by a querier-provided nonce (see [90]), which prevents both the incorporation of old validity information and replay attacks. This implies that f was valid during the protocol execution, which is a contradiction. □

The fact that existing distributed proof construction protocols use incrementally consistent views when making authorization decisions is exactly what leads to the types of safety violations discussed in Section 4.1. This is because incremental consistency provides no guarantees regarding the overlap of the observed validity periods for facts whose state is stored in V in the event that *any* fact used during the

proof construction is not a stable assertion. The other consistency levels defined in this section will address this problem.

Query Consistency

The next more stringent level of consistency that we define is query consistency. Informally, this consistency level guarantees that all facts used to construct a distributed proof were valid simultaneously at the time that the query triggering that proof construction was issued. We formally define query consistency in terms of the predicate $\phi_{query} : 2^{\mathcal{S}} \times T \rightarrow \mathbb{B}$, as follows:

$$\begin{aligned} \phi_{query}(V, t_{iss}) \quad \equiv \quad & \forall s \in V : concrete(s) \\ & \wedge (s.t_{\alpha} \leq t_{iss} \leq s.t_{\omega}) \end{aligned} \tag{4.5}$$

DEFINITION 4.3.9 (Query Consistency). *A view V is* query consistent *with respect to a query issued at time t_{iss} if and only if $\phi_{query}(V, t_{iss})$ is* true.

If an authorization policy is satisfied using a query consistent view, the semantics of policy satisfaction in the distributed proof construction setting remain the same as if the proof had been constructed using a centralized proof framework supporting transactional evaluation (e.g., a Prolog theorem prover). In the event that any facts necessary to construct a given proof of authorization are unstable (i.e., their value can change once set), a view consistency level that is at least as strong as query consistency should be enforced to ensure that the satisfaction of a given authorization policy carries the same meaning as policy writers and analysts would expect it to have. In this respect, query consistency bears many similarities to the internal and endpoint consistency levels introduced in Chapter 3.

Interval Consistency

The most stringent consistency level that we consider in this chapter is called interval consistency. We say that some view V is interval consistent during some interval $[t_s, t_e]$ if each fact state tuple in V encodes a concrete validity interval that includes at least $[t_s, t_e]$. More formally, we define interval consistency using the predicate $\phi_{interval} : 2^{\mathcal{S}} \times T \times T \rightarrow \mathbb{B}$, as follows:

$$\phi_{interval}(V, t_s, t_e) \equiv \forall s \in V : concrete(s) \wedge (s.t_\alpha \leq t_s \leq t_e \leq s.t_\omega) \quad (4.6)$$

DEFINITION 4.3.10 (Interval Consistency). *A view V is* interval consistent *for a time interval $[t_s, t_e]$ if and only if $\phi_{interval}(V, t_s, t_e)$ is* true.

The above definition of interval consistency is a reformulation of the definition of interval consistency developed in Chapter 3, altered to fit within the formalization of the consistency problem presented in Section 4.3.2. In distributed proving, the notion of interval consistency is useful for two primary reasons. First and foremost, interval consistency is important in the event that a resource provider wishes to monitor the conditions that lead to the permission of a given resource access. For instance, the hospital smart room discussed in Section 4.1 may wish to first check that Alice is the only person located in her locked office before allowing her to project patient records onto the wall and then continue to monitor these conditions. If her door subsequently became unlocked, for instance, access to the projector could be revoked.

At the implementation level, interval consistency can also be useful in the event that a proof tree is constructed that permits access to a given resource, but that view cannot be shown to be query consistent. The fact that a proof could be formed at all implies that it is possible that the facts that make up the proof were valid simultaneously, even though this could not be guaranteed from the view used to construct the proof. If it is faster to recheck a proof than it would be to generate the proof tree again, then this recheck could lead to an interval consistent view during the interval $[t_{rcv}, t_{recheck}]$, where t_{rcv} is the time that the original proof was returned to the resource provider and $t_{recheck}$ is the time at which the resource provider begins revalidation of the proof tree. We will explore this case further in Section 4.4.3.

Sliding Windows of Consistency

Although powerful, the notion of interval consistency defined above may be too strong in some circumstances. For example, in pervasive computing environments with rapid contextual changes, fact validity statuses may fluctuate often around some acceptable baseline; this could lead to situations in which views were repeatedly determined to be inconsistent and cause numerous service interruptions (e.g., consider a policy that is in some way predicated on the number of occupants in a

busy hallway). Rather than falling back on the notion of query consistency, which provides no continuing validity checks, entities may wish to enforce a level of view consistency that provides guarantees somewhere between what is afforded by the query and interval consistency levels.

As one interesting example, a service provider might wish to enforce the constraint that at each time t, all facts used to justify resource access must have been simultaneously valid at some time t' such that $t - \Delta \leq t' \leq t$, where Δ is the length of a sliding window defined by the service provider. More formally, we define this notion of sliding windows of consistency using the predicate $\phi_{sliding} : 2^{\mathcal{S}} \times T \times T \times \mathbb{N} \rightarrow \mathbb{B}$ as follows:

$$\begin{aligned} \phi_{sliding}(V, t_s, t_e, \Delta) \quad \equiv \quad & \forall t \in [t_s, t_e]\ \exists t' \in [t - \Delta, t] \text{ such that} \\ & \forall s \in V_{t'} : concrete(s) \wedge (s.t_\alpha \leq t' \leq s.t_\omega) \end{aligned} \tag{4.7}$$

DEFINITION 4.3.11 (Sliding Window Consistency). *A view V is* sliding window consistent *for a time interval $[t_s, t_e]$ and a window size Δ if and only if the predicate $\phi_{sliding}(V, t_s, t_e, \Delta)$ is* true.

The notion of sliding window consistency is only useful in situations where the validity of the facts being monitored in a particular view change over time. As a result, sliding window consistency is not a property of a single view, but is rather a property of a *series* of views representing the observed fluctuations of a set of facts. The definition of Equation 4.7 reflects this by introducing the notion of a *subscripted view*. The view $V_{t'}$ used in this definition refers to the instance of the view V containing information about fact validity statuses at the time t'. In Section 4.4.4, we will see that the algorithm used to enforce this notion of view consistency requires periodic evaluation of intermediate conditions, which is a reflection of this notion of evolving views.

4.4 Algorithm Details

In this section, we discuss modifications to the Minami-Kotz distributed proof construction algorithm that ensure the use of consistent system views during policy evaluation. As this algorithm trivially ensures that an incrementally consistent view is used (by Theorem 4.3.8), we will focus our discussion on creating query, interval, and sliding window consistent views.

Algorithm 4.1 A query consistency enforcement algorithm

```
1: // Receive a fact response tuple relevant to a query issued at time t_iss
2: // from some entity e. Only invoked on true facts.
3: Function RcvFact(f ∈ F, d ∈ ℕ, e ∈ E, t_iss ∈ T, V ∈ 2^S)
4: t_rcv ← NOW
5: if t_rcv − t_iss ≤ d(1 − δ) then
6:     V.insert(Encode(f), e, t_iss, t_iss, false)
7: else
8:     V.insert(Encode(f), e, t_iss, NOW, true)
9:
10: // Check the query consistency condition on a view V relative
11: // to a query issued at time t_iss.
12: Function CheckQuery(V ∈ 2^S, t_iss ∈ T)
13: for all s ∈ V do
14:     if s.fuzzy ∨ (t_iss < s.t_α) ∨ (s.t_ω < t_iss) then
15:         return false
16: return true
```

4.4.1 Preliminaries

In this section, we will be concerned with both the correctness and security properties of our proposed proof construction algorithm modifications. In addition to proving the soundness of our consistency enforcement algorithms, we will also address their proximity to *ideal completeness*. As in Chapter 3, a ϕ-consistency enforcement algorithm is said to be ideally complete if and only if it is capable of constructing ϕ-consistent views for all protocol executions in which an ideal algorithm run by an omniscient entity could construct a ϕ-consistent view. Further, we will ensure that each proposed modification is a *policy-safe modification* to the proof construction protocol. That is, we will show that our modifications do not violate the integrity or confidentiality policies specified by each entity.

4.4.2 Query Consistency

We now show that with relatively minor changes, the Minami-Kotz distributed proof construction protocol can be modified to use query consistent views when making authorization decisions. As presented by Minami and Kotz [90], this proof construction algorithm assumes that each knowledge base KB is defined as a subset of all possible facts, $\mathcal{F}$. Rather, we will define a knowledge base KB as a subset of $\mathcal{F} \times T$ in which each fact is associated with the local time at which it was inserted into KB. This allows each node to track the duration of a given fact's validity locally.

To leverage this new knowledge base format, the format of query responses must also be altered. Rather than an entity e responding to some query $?f$ with a Boolean response $b \in \mathbb{B}$ indicating whether f is considered valid by e (as in Section 4.2), they will instead respond with a *fact response tuple* of the form $\langle b, d \rangle \in \mathbb{B} \times \mathbb{N}$. The b

component of this tuple indicates whether e considers f to be valid, as before, and the d component of this tuple represents the length of time that e acknowledges that f has been true, or some duration less than this if the exact duration of validity is considered sensitive. In the event that f is a base atom, d is (at most) the difference between the current time and the time associated with f in e's knowledge base; if f is the head of a Horn clause f :- $f_1, \ldots, f_n$, then d is set to be (at most) the minimum such duration associated with any of $f_1, \ldots, f_n$. In the case that f is false, d is set to 0. Note that neither f nor any $f_1, \ldots, f_n$ need to be locally-stored facts.

Given the above modifications to the formats of entities' knowledge bases and query responses, we now present the details of Algorithm 4.1, which facilitates the creation of query consistent views. In Algorithm 4.1 and all other algorithms presented in this chapter, we make the following assumptions:

- The current local time is available via the local variable NOW.
- The absolute value of the maximum clock drift rate between any two entities in the system is no more than some constant δ. This does not imply that clocks are in any way synchronized, only that for each n time units that pass one entity, no less than $n(1-\delta)$ time units and no more than $n(1+\delta)$ time units pass at any other entity.
- Fact state tuples can be inserted into a view data structure via the function $insert : \{0,1\}^\ell \times \mathcal{E} \times T \times T \times \mathbb{B} \rightarrow \perp$. Note, for instance, that $V.insert(id, e, t, t', b)$ will replace any existing fact state tuples in V that have the identifier id and were received from entity e (see Definition 4.3.5).
- The function ENCODE : $\mathcal{F} \rightarrow \{0,1\}^\ell$ returns an encoding of some fact $f \in \mathcal{F}$ suitable for insertion into a view (see Definition 4.3.4).
- Each entity detects the failure of other participants in the algorithm by using timeouts. If the initiator of an algorithm detects such a failure, the algorithm can be aborted and restarted. To simplify the presentation of our algorithms, further details of this process are omitted.

Algorithm 4.1 consists of two functions that are to be used by the querying entity. Whenever the querier issues a new query, she records the query issue time t_{iss} and chooses a view V to which the results of her query are considered relevant; V need not be a new empty view. In the event that the response to her query is true, Alice invokes the RCVFACT function. If the fact provider attests that the fact whose status was queried was valid for some duration d that is longer than the time between when the query was issued and when its response was received, this function inserts a fact state tuple into V asserting that the corresponding fact was

valid at time t_{iss}. Otherwise, a fact state tuple encoding a fuzzy interval is inserted into V. The function CHECKQUERY checks to see that ϕ_{query} is true, as defined by Equation 4.5.

THEOREM 4.4.1. *If the function* CHECKQUERY(V, t_{iss}) *returns* ***true****, then V is query consistent relative to the query issue time t_{iss}, provided that V was constructed using only the* RCVFACT *function.*

Proof. The CHECKQUERY function clearly enforces the constraint that all fact state records encode concrete validity intervals that include the time t_{iss}. Thus CHECKQUERY$(V, t_{iss}) \leftrightarrow \phi_{query}(V, t_{iss})$ and V is query consistent with respect to the time t_{iss} by Definition 4.3.9, provided that all concrete validity intervals established by RCVFACT are correct. Lines 5 through 8 of Algorithm 4.1 ensure that RCVFACT inserts a concrete validity interval into V if and only if the validity duration, d, reported by the fact provider is longer than the query round trip time, even when adjusted to assume the largest possible clock drift between entities. Since the query and its associated response can be causally linked by nonces used in the underlying proof construction protocol (see Minami and Kotz [90]), we can infer that the fact provider sent its response to the query at some time $t \geq t_{iss}$. This implies that the fact associated with the state tuple being inserted into V was valid at t_{iss} because $t - d(1 - \delta) \leq t_{iss} \leq t$ for all possible values of t such that $t_{iss} \leq t \leq t_{rcv}$ since $t_{rcv} - t_{iss} \leq d(1 - \delta)$. □

THEOREM 4.4.2. *Algorithm 4.1 is a policy-safe modification to the Minami-Kotz distributed proof construction protocol.*

Proof. Minami and Kotz [90] prove that their distributed proof construction algorithm constructs a proof of authorization only if the integrity policies of every participating entity are satisfied. Since Algorithm 4.1 does not alter the mechanism through with the proof construction algorithm constructs proof trees, it does not affect the enforcement of any entity's integrity policies. We now show that Algorithm 4.1 does not affect the enforcement of any entity's confidentiality policies.

Minami and Kotz [90] also show that their distributed proof construction algorithm constructs proof trees only if the confidentiality policies of each participating entity are satisfied. Recall that these confidentiality policies are enforced by optionally encrypting query responses of form $b \in \mathbb{B}$ using a key bound to some entity e higher up the proof tree than the direct querying entity (see Section 4.2.3). Since the modified query responses of form $\langle b, d \rangle \in \mathbb{B} \times \mathbb{N}$ returned by a fact provider using Algorithm 4.1 can be encrypted in this same manner, each entity's confidentiality policies are still enforced. Therefore, Algorithm 4.1 makes only policy-safe

modifications to the underlying distributed proof construction protocol. □

Although Algorithm 4.1 is sound (by Theorem 4.4.1), it is not ideally complete. It could be the case that a certain fact was valid at the time that a query was issued, even if the validity duration reported by the fact provider is less than the query round-trip time; this is an inevitable consequence of the use of a causal ordering rather than synchronized clocks. Although not a violation of ideal completeness, this algorithm can also fail in the event that the validity of some fact is not monitored until the first query regarding this fact is issued. Both of these cases make it desirable to have an efficient means of revalidating a given view, as it is likely to be found consistent if rechecked. This leads directly to the stronger notion of interval consistency.

4.4.3 Interval Consistency

Recall from Chapter 3 that establishing an interval consistent view typically involves observing that the validity statuses of the facts comprising the view do not change or fluctuate during the course of several observations of portions of the system. In the distributed proving setting, one cannot simply construct a given proof twice to establish an interval of validity, as there would be no guarantee that the values of facts did not fluctuate between proofs or even that the same proof tree was generated for each query.[2] The requery method can succeed, however, if we leverage caches at intermediate nodes to ensure that the same proof tree is constructed at each invocation and that fluctuations can be detected (via cache misses caused by proactive revocations). The intermediate node caches proposed by Minami and Kotz [91] could be modified to suit this purpose.

Although this modified requery strategy for ensuring interval consistent views is appealing due to its simplicity, it is in fact a worst-case strategy for a number of reasons. First, this strategy requires excessive storage of data at intermediate nodes which is undesirable if nodes wish to remain autonomous. Second, the requery strategy requires that the entire proof tree be traversed twice, even though only the values managed by the leaves of the proof tree are of any significance to whether the proof succeeds; this results in high communication overheads. Lastly, due to reliance on intermediate node caches, a failure of *any* node contributing to the proof tree can cause the revalidation process to fail.

Assuming that the rules of inference dictating the structure of a proof tree are not

[2] Recall that the portions of the proof tree outside of the querier's integrity policies are unknown to the querier; these portions of the proof tree may differ between invocations and go undetected.

revoked over time, we can design a more optimal strategy for ensuring interval consistent views. Specifically, we can alter the proof construction protocol in such a way as that the querying entity would learn not only whether a proof succeeded, but also a set of *association tuples*, each of which binds the identity of some leaf entity e in the proof tree to a fact identifier that can be used to recheck the status of the fact provided by e. This strategy eliminates the need for intermediate node caches, incurs the lowest possible overall communication overheads during a proof recheck, and fails only if a data-providing entity fails. Even though this leaf exposure strategy is optimal in many respects, it can potentially violate the confidentiality policies of the leaf entities.

In an attempt to balance the efficiency of the leaf exposure strategy with the privacy preservation of the requery strategy, we propose the *leaf indirection strategy* for constructing interval consistent views. As was the case with the query enforcement strategy presented in Section 4.4.2, we require slight modifications to formats of the entities' knowledge bases and query responses.[3] We will define a knowledge base KB as a subset of $\mathcal{F} \times \{0,1\}^\ell$ in which facts are associated with some locally-unique identifier. It is important that each time a fact is inserted into a knowledge base, it is associated with a previously-unused local identifier in $\{0,1\}^\ell$. Further, an entity e will respond to a query of the form $?f$ with a response tuple of the form $\langle b, (\langle e', id \rangle)_{K_q} \rangle \in \mathbb{B} \times \{0,1\}^n$. The b component of this tuple indicates whether e considers f to be valid, as in the unmodified proof system. The e' and id components of this tuple form an association tuple from the set $\mathcal{E} \times \{0,1\}^\ell$, as described above, though this association tuple is encrypted with the public key of the original querying entity, q. As in the leaf exposure strategy, e may choose to bind herself to the proof tree, in which case $e' = e$ and id is set to the identifier currently associated with f in e's knowledge base. However, e can instead choose a trusted *indirect entity*, ie, at random, obtain a nonce n from ie, and bind ie to the proof tree by setting $e' = ie$ and $id = n$. Each indirect entity maintains a small *remote cache* that associates locally-chosen nonces with ⟨entity, fact identifier⟩ pairs to facilitate the proof recheck process.

The leaf-indirection strategy for constructing interval consistent views is implemented by Algorithm 4.2. Prior to explaining this algorithm in detail, we first assume that entities have access to the following local data structures and methods, in addition to those required in Section 4.4.2:

[3] Although the modifications required for the leaf indirection strategy are presented independently of the modifications required for query consistency, this need not be the case. In practice, both sets of modifications can be used together to allow for the creation of either query or interval consistent views.

- Each entity maintains a set of locally-trusted indirect entities, $Indirect$.
- The symbol $\leftarrow_r$ denotes random assignment from some set. For example, $e \leftarrow_r Indirect$ chooses a random member from the set of trusted indirect entities.
- An entity's node identifier can be accessed via the local variable ME.
- The function GETFRESHNONCE: $\perp \rightarrow \{0,1\}^\ell$ chooses a previously unused identifier to be associated with some fact or fact provider.
- The local knowledge base is accessible via the data structure KB. The function $KB.contains : \{0,1\}^\ell \rightarrow \mathbb{B}$ checks whether the fact associated with a given identifier is currently in the local knowledge base.
- An entity's remote cache is accessible via the $RemoteCache$ data structure. This data structure has member functions $insert : \{0,1\}^\ell \times \mathcal{E} \times \{0,1\}^\ell \rightarrow \perp$, $contains : \{0,1\}^\ell \rightarrow \mathbb{B}$, $lookup : \{0,1\}^\ell \rightarrow \mathcal{E} \times \{0,1\}^\ell$, and $delete : \{0,1\}^\ell \rightarrow \perp$.

Algorithm 4.2 works as follows. An entity contributing a base fact to some proof tree invokes the GENERATEASSOCIATION function to generate the association tuple that will be propagated back up the proof tree to the initial querier. If the entity q for whom this association tuple is being prepared is authorized by the local entity's confidentiality policies to learn the value of the fact associated with id, this function binds the local entity to the provided fact identifier. If q is not authorized to learn of the local entity's involvement in the proof process, a randomly-chosen trusted indirect entity is bound to the proof tree via a call to the INSERTREMOTE(e, id) function. This function triggers the execution of the INSERTASSOCIATION function at the entity e; INSERTASSOCIATION chooses a fresh nonce and binds this to the pair $\langle e', id \rangle$, where e' is the entity who called INSERTREMOTE(e, id). The nonce is then returned to the entity e'. The initial querier thus receives both the proof tree constructed by the algorithm discussed in Section 4.2 and a list of association tuples binding either leaf entities or indirect leaf entities to the proof tree. Each of these association tuples is used to construct fact state tuples in some view V by using the RCVASSOC function.

Once the view V contains all of the necessary fact state tuples, the initial querier can then attempt to establish an interval of consistency through one or more calls to the RECHECKVIEW(V) function. For each fact state tuple $s \in V$, this function uses the ASKREMOTE function to query the remote entity $s.e$ to see if the fact associated with $s.id$ is still valid. If the fact associated with s is still valid, then the validity interval in s is updated. Fuzzy validity intervals are turned into concrete validity intervals ranging from the end of the fuzzy interval until the time that the recheck

Algorithm 4.2 An interval consistency enforcement algorithm

```
1: // Generate an association tuple for the fact associated with identifier id
2: // to be sent to the initial querying entity q
3: Function GENERATEASSOCIATION(id ∈ {0,1}^ℓ, q ∈ ℰ)
4: if q not authorized to learn fact associated with id then
5:     e ←_r Indirect
6:     id′ ← INSERTREMOTE(e, id)
7:     return ⟨e, id′⟩
8: else
9:     return ⟨ME, id⟩
10:
11: // Accepts an entry to the RemoteCache table after
12: // entity e calls INSERTREMOTE
13: Function INSERTASSOCIATION(e ∈ ℰ, id ∈ {0,1}^ℓ)
14: id′ ← GETFRESHNONCE()
15: RemoteCache.insert(id′, e, id)
16: return id′
17:
18: // Insert a fact state tuple associated with the association record ⟨e, id⟩
19: // bound to a query issued at time t_iss into the view V.
20: Function RCVASSOC(⟨e, id⟩ ∈ ℰ × {0,1}^ℓ, t_iss ∈ T, V ∈ 2^𝒮)
21: V.insert(id, e, t_iss, NOW, true)
22:
23: // Recheck the fact tuples making up a view V
24: Function RECHECKVIEW(V ∈ 2^𝒮)
25: for all s ∈ V do
26:     t ← NOW
27:     if ASKREMOTE(s.id, s.e) then
28:         if s.fuzzy then
29:             V.insert(s.id, s.e, s.t_ω, t, false)
30:         else
31:             V.insert(s.id, s.e, s.t_α, t, false)
32:
33: // Revalidate the fact identified by id
34: Function RECHECKFACT(id ∈ {0,1}^ℓ)
35: if KB.contains(id) then
36:     return true
37: if RemoteCache.contains(id) then
38:     ⟨e, id′⟩ ← RemoteCache.lookup(id)
39:     b ← ASKREMOTE(e, id′)
40:     if ¬b then
41:         RemoteCache.delete(id)
42:     return b
43: else
44:     return false
45:
46: // Check the interval consistency condition on a view V relative
47: // to the time interval [t_s, t_e]
48: Function CHECKINTERVAL(V ∈ 2^𝒮, t_s ∈ T, t_e ∈ T)
49: for all s ∈ V do
50:     if s.fuzzy ∨ (t_s < s.t_α) ∨ (s.t_ω < t_e) then
51:         return false
52: return true
```

was invoked; concrete validity intervals are just extended. The ASKREMOTE(id, e) function works by invoking the RECHECKFACT(id) function at the entity e. This function returns true if the fact associated with id is still in e's local knowledge base or in the knowledge base of the entity associated with the nonce id in e's remote cache, and returns false otherwise. The function CHECKINTERVAL(V) checks to see that $\phi_{interval}(V)$ holds, as defined by Equation 4.6.

THEOREM 4.4.3. *If the function* CHECKINTERVAL(V, t_s, t_e) *returns* true*, then* V *is interval consistent on the interval* $[t_s, t_e]$ *provided that* V *was constructed using only calls to the* RCVASSOC *and* RECHECKVIEW *functions.*

Proof. The CHECKINTERVAL(V, t_s, t_e) function enforces the constraint that all fact state tuples in V encode concrete validity intervals that include at least the interval $[t_s, t_e]$. Therefore, CHECKINTERVAL$(V, t_s, t_e) \leftrightarrow \phi_{interval}(V, t_s, t_e)$, which implies that V is interval consistent on the interval $[t_s, t_e]$ by Definition 4.3.10, provided that all concrete validity intervals established by RCVASSOC and RECHECKVIEW are correct. RCVASSOC$(\langle e, id\rangle, t_{iss})$ inserts a fuzzy validity interval bounded by the query issue time, t_{iss}, and the association tuple receipt time for the fact f described by identifier id. This is a legitimate action, as nonces used by the underlying proof construction protocol (see [90]) allow us to causally link the query and its response and therefore establish that the fact provider asserted f's validity at some time $t \geq t_{iss}$. We must now show that RECHECKVIEW updates these fuzzy intervals correctly.

Assuming that the ASKREMOTE function correctly determines whether a given fact is still true at the providing entity, RECHECKVIEW extends the validity interval for each fact state tuple whose corresponding fact is still valid. Assume that the recheck process for a fact state tuple s corresponding to some fact f starts when $NOW = t_r$ and succeeds. If s encodes the fuzzy validity interval $[s.t_\alpha, s.t_\omega]$, s is updated to encode the concrete validity interval $[s.t_\omega, t_r]$, since f was true at some time $t \leq s.t_\omega$ and was not yet revoked at some later time $t' \geq t_r$. If s encodes a concrete validity interval $[s.t_\alpha, s.t_\omega]$, it is extended to encode the concrete validity interval $[s.t_\alpha, t_r]$. We now show that RECHECKFACT(id)—which is invoked at entity e by the call ASKREMOTE(id, e)—correctly assesses the validity of the fact associated with the identifier id.

We must consider both the case in which the fact f associated with the identifier id was originally stored in e's local knowledge base and the case in which id was an entry in e's remote cache. In the case where f was stored in e's local knowledge base, line 35 of Algorithm 4.2 returns true if e's knowledge base still contains the fact f associated with id; we know that f has not yet been revoked because the GETFRESHNONCE function ensures that fact identifiers are not reused. If f has since been removed from e's knowledge base, then the call to $KB.contains(id)$ will fail. The check on line 37 will then fail because the state tuple associated with id was originally stored locally and thus would not be associated by GETFRESHNONCE with an entry in e's remote cache. This failure would then cause RECHECKFACT to return false, which implies that RECHECKFACT performs correctly in the case

in which the fact f associated with identifier id was originally stored in e's local knowledge base.

We now consider the case in which id was originally associated with an entry in e's remote cache. Line 38 first determines the tuple $\langle e', id' \rangle$ associated with the identifier id in e's remote cache. If this lookup fails, we know that the fact that was indirectly associated with the identifier id has been revoked, as entries in e's remote cache are only removed after failed lookups (by line 41). By reasoning similar to that used above, the call to AskRemote on line 39 will cause RecheckFact to return true in the event that the fact f' associated with id' in e''s knowledge base has not yet been revoked. Again, we know that this is not a false positive, as GetFreshNonce ensures that fact identifiers are used at most once. If f' has been removed from e''s knowledge base, but not from e's remote cache, this call to AskRemote will return false. This will cause the entry associated with id to be removed from e's remote cache; RecheckFact will then return false, as expected.

The fact that RecheckFact behaves as expected implies that AskRemote correctly assesses the continuing validity of remotely store facts. This, in turn, implies that RecheckView correctly updates the validity intervals encoded in the fact state tuples of a given view initially constructed by one or more calls to the RcvAssoc function. Since views can be correctly constructed by calls to the RcvAssoc and RecheckView functions and we have shown that CheckInterval$(V, t_s, t_e) \leftrightarrow \phi_{interval}(V, t_s, t_e)$, we can conclude that CheckInterval(V, t_s, t_e) returns true if and only if V is interval consistent on the interval $[t_s, t_e]$. □

Although Algorithm 4.2 is shown to be sound by Theorem 4.4.3, it is not ideally complete. That is, an omniscient entity may have been able to observe an interval consistent view even if Algorithm 4.2 fails. This can occur because the set of rechecks initiated after the time t_e takes a non-zero amount of time to complete. As with the completeness limitations discussed in Section 4.4.2, this is an artifact of relying on causal orderings to establish validity intervals, rather than perfectly synchronized clocks. We now show that Algorithm 4.2 is a policy-safe modification to the underlying distributed proof construction protocol.

Theorem 4.4.4. *Algorithm 4.2 is a policy-safe modification to the Minami-Kotz distributed proof construction protocol.*

Proof. As was the case with Algorithm 4.1, Algorithm 4.2 does not affect the construction of distributed proof trees. Therefore, the proof given by Minami and Kotz [90] stating that proof trees are constructed only if the integrity policies specified by each participating entity are respected still holds. We must now show that

the confidentiality policies specified by each entity are still respected. To this end, we must show that no unauthorized entity along the path from the querier q to some fact provider e can learn both the fact f provided by e and f's validity as reported by e. To address the most general case, we will assume that learning e's identity is sufficient for an unauthorized entity to infer f. We then show that (i) each entity participating in the construction of the proof tree that is not entitled to know f cannot learn e's identity, and (ii) entities that do know f but should not learn f's validity cannot infer it.

We first treat case (i) and show that each unauthorized entity u in the proof tree that should not learn f does not learn e's identity. There are two sub-cases: $u = q$ and $u \neq q$. Consider the case where u is an *intermediate entity* in the proof tree; that is, $u \neq q$. In this case, u cannot learn e's identity, as the association tuple that might possibly bind e to the proof tree is encrypted with q's public key, K_q. In the case that $u = q$, we must show that u cannot learn e's identity. In this case, q receives an association tuple binding an indirect entity ie to the fact provided by e. Since ie is trusted by e not to reveal e's identity, q cannot learn the identity of e and thus cannot infer the hidden fact f provided by e.

Case (ii) is handled by the encryption of sensitive query responses as described in Section 4.2.3. Since the incorporation of association tuples into query responses does not affect this encryption process, the proof construction algorithm will correctly enforce the confidentiality of responses as proven by Minami and Kotz [90]. As we have shown that each entity that should not learn the fact f provided by e cannot learn f and that entities that should not learn f's validity cannot learn it, we can conclude that e's confidentiality policies are correctly enforced. Since both e's confidentiality policies and integrity policies are correctly enforced, we can conclude that Algorithm 4.2 is a policy-safe modification to the underlying proof system. □

Although Algorithm 4.2 is a policy-safe modification to the Minami-Kotz distributed proof construction framework, it does nonetheless reveal additional information to the initial querying entity q. Specifically, in addition to knowing the portion of the initial proof tree specified by its integrity policies, q learns a list of association tuples declaring certain entities to be (possibly indirect) contributors of atomic facts to the proof tree. We now prove that this list of association tuples gives q only minimal information regarding the structure of the generated proof tree beyond what is implied by its integrity policies.

THEOREM 4.4.5. *Given the set of association tuples $\mathcal{A} = \{\langle e_1, n_1\rangle, \ldots, \langle e_i, n_i\rangle\}$ associated with a given proof tree, the initial querier q learns only the number of entities con-*

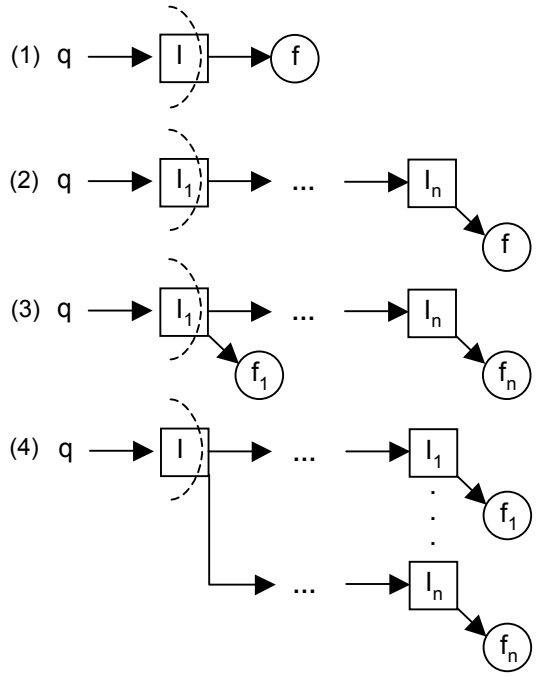

Figure 4.5: A diagram illustrating several possible proof tree structures. Inference nodes are represented by squares and fact leaves are represented by circles. The dashed lines indicate the expected leaves of the proof tree as perceived by the querier.

tributing facts to the proof tree and, in some cases, whether the proof tree extends beyond the proof tree implied by their integrity policies.

Proof. Assume without loss of generality that each entity involved in the construction of a distributed proof makes at most one inference step. Let the set E contain the leaf entities implied by q's integrity policies, henceforth called the set of *expected leaves* of the proof tree. To prove this claim, we must explore two cases: $|E| = |\mathcal{A}|$, and $|E| < |\mathcal{A}|$. If $|E| = |\mathcal{A}|$, we know that each expected leaf node $e \in E$ is either a fact provider or the initiator of a chain of inference forming a linear subproof whose leaf provides a fact to the proof generated by q's query. If e is mentioned explicitly in an association tuple $a \in \mathcal{A}$, then q cannot conclude whether e contributed directly to the proof tree or was chosen as an indirect entity for the actual leaf, e', of the linear subproof initiated by e. These two indistinguishable sub-cases are shown in Figure 4.5 parts (1) and (2). Note that in Figure 4.5, each inference node I_k is associated with a unique entity e_k by our prior assumption. If e is *not* explicitly mentioned in any association tuple in $\mathcal{A}$, then q cannot differentiate between the case in which e contributed a fact to the proof tree via an indirect entity and the case in which e initiated a chain of inference resulting in a linear subproof.

If $|E| < |\mathcal{A}|$, then q knows that the proof tree certainly extends beyond the proof tree implied by her integrity policies and that at least one $e \in E$ initiated a subproof contributing multiple facts to the proof. In the event that $|E| > 1$, q cannot infer which entity or entities in E initiated subproofs contributing multiple facts to the proof, as the set of association tuples $\mathcal{A}$ encodes no structural information. If $|E| = 1$, then q clearly knows which entity was responsible for initiating a subproof that contributed multiple facts to the proof tree. However, q cannot differentiate between linear and branching chains of inference, again, as no structural information is encoded in the set $\mathcal{A}$. This case is illustrated in Figure 4.5 parts (3) and (4). This shows that q learns no information beyond the number of facts used in the proof tree and whether, in some cases, the proof tree extends beyond the expected leaves of the proof tree. □

4.4.4 Sliding Windows of Consistency

Recall from Section 4.3.3 that the definition of sliding window consistency requires the use of a *sequence* of views recording observational information regarding the validity of the facts used during the construction of a given distributed proof. As a result, the algorithm used to enforce this consistency condition cannot follow the "construct and validate" strategies used to enforce the notions of interval and query consistency. Rather, algorithms for enforcing sliding window consistency constraints must repeatedly sample the state of the proof tree, evaluating intermediate consistency conditions all the while.

A naive approach to enforcing this consistency condition involves repeated use of Algorithm 4.1. If a query evaluator uses this algorithm to establish a query consistent view every τ time units for some $\tau \leq \Delta$, it is easy to show that the $\phi_{sliding}$ predicate (Equation 4.7) is satisfied. However, as we will see in Section 4.5, the overhead of constructing an initial proof tree (e.g., as in Algorithm 4.1) is far greater than the cost of rechecking an existing proof tree. This implies that this naive approach to enforcing the sliding window consistency condition is computationally more expensive than is strictly necessary. A more cost-effective alternative is to create an algorithm that leverages both the simplicity of Algorithm 4.1 and the efficient rechecking process used by Algorithm 4.2.

To accomplish this, we require that knowledge bases are of the form $KB \subseteq \mathcal{F} \times T \times \{0,1\}^{\ell}$. This allows each fact in an entity's knowledge base to be associated with a period of validity (as in Section 4.4.2) and a locally-unique identifier (as in Section 4.4.3). Unlike in Section 4.4.3, however, the unique identifier for a given fact does not need to change over time. An entity e will respond to a query of the

form $?f$ with a response tuple of the form $(b, d, (\langle e', id \rangle)_{K_q}) \in \mathbb{B} \times \mathbb{N} \times \{0,1\}^n$. As before, the b component of this tuple indicates whether e considers f to be valid, and the d component represents the duration of validity recorded for f at the time that the response tuple was generated. As in Section 4.4.3, the e' and id portions of this response form an association tuple that provides a means for the querier to recheck the validity of a given fact without revealing which fact is being rechecked or which entity maintains the status information for that fact. After carrying out its initial query, the querier will then be able to use the set of association tuples that it has gathered to recheck the simultaneous validity of the facts in the view much more efficiently than is possible through repeated use of Algorithm 4.1.

The above means of enforcing the sliding window consistency condition is implemented by Algorithm 4.3. Prior to explaining this algorithm in detail, we make the following assumptions, in addition to those made in Section 4.4.2:

- The local knowledge base is accessible via the the data structure KB. The function $KB.isLocalId : \{0,1\}^\ell \rightarrow \mathbb{B}$ checks whether the fact associated with a given identifier has ever been monitored by this knowledge base. The function $KB.lookup : \{0,1\}^\ell \rightarrow \mathbb{B} \times T$ returns the validity status and duration of validity for the fact associated with a given identifier.
- An entity's remote cache is accessible via the $RemoteCache$ data structure. This data structure has member functions $contains : \{0,1\}^\ell \rightarrow \mathbb{B}$ and $lookup : \{0,1\}^\ell \rightarrow \mathcal{E} \times \{0,1\}^\ell$.

Algorithm 4.3 works as follows. As described earlier, when an entity contributes a fact to a proof tree, they return the fact's validity status, a duration of validity, and an association tuple describing how the validity status of this fact can be efficiently rechecked. For brevity, the details of managing the remote cache that manages these associate tuples have been omitted from Algorithm 4.3, but the procedure for choosing indirect nodes and creating association tuples used in Algorithm 4.2 could be used in this algorithm without modification. After the querying node receives a response tuple, they use RCVRESPONSE method to insert this response tuple into their local view. This process represents the initialization phase of the sliding window consistency enforcement algorithm. We now describe the ongoing verification phase of this algorithm.

To verify the consistency of a view V constructed using calls to the RCVRESPONSE method, the querier calls the CHECKSLIDING function, providing as arguments the view V, a window length Δ, and the time at which V was last verified to be sliding window consistent. This function first saves the current time and then calls the RECHECKVIEW method. RECHECKVIEW checks the current validity status of

Algorithm 4.3 A sliding window consistency enforcement algorithm

```
1: // Receive a fact response tuple relevant to a query issued at time t_iss
2: // from some entity e. Only invoked on true facts.
3: Function RcvResponse(⟨e, id⟩ ∈ E × {0,1}^ℓ, d ∈ ℕ, t_iss ∈ T, V ∈ 2^S)
4: t_rcv ← NOW
5: if t_rcv − t_iss ≤ d(1 − δ) then
6:     V.insert(id, e, t_iss, t_iss, false)
7: else
8:     V.insert(id, e, t_iss, NOW, true)
9:
10: // Recheck the fact tuples making up a view V
11: Function RecheckView(V ∈ 2^S, t ∈ T)
12: for all s ∈ V do
13:     (b, d) ← AskRemote(s.id, s.e)
14:     ts ← NOW
15:     if b then
16:         if ¬s.fuzzy ∧ (ts − s.t_ω ≤ d(1 − δ)) then
17:             V.insert(s.id, s.e, s.t_α, t, false)
18:         else if ts − t ≤ d(1 − δ) then
19:             V.insert(s.id, s.e, t, t, false)
20:
21: // Revalidate the fact identified by id
22: Function RecheckFact(id ∈ {0,1}^ℓ)
23: if KB.isLocalId(id) then
24:     return KB.lookup(id)
25: else if RemoteCache.contains(id) then
26:     ⟨e, id′⟩ ← RemoteCache.lookup(id)
27:     return AskRemote(e, id′)
28: else
29:     ERROR
30:
31: // Check the sliding window consistency condition on a view V relative to the window size Δ.
32: // The parameter t_last represents the last time that this consistency condition was true. The
33: // second component in the tuple returned by this function indicates the last time at which the
34: // sliding window consistency condition was verified to hold. This can, in turn, be used as
35: // the t_last argument to future invocations of this function.
36: Function CheckSliding(V ∈ 2^S, Δ ∈ ℕ, t_last ∈ T)
37: t ← NOW
38: if (t_last > 0) ∧ (t_last < t − Δ) then
39:     return (false, t_last)
40: RecheckView(V, t)
41: status ← true
42: for all s ∈ V do
43:     if s.fuzzy ∨ (t < s.t_α) ∨ (s.t_ω < t) then
44:         status ← false
45: if status then
46:     return (true, t)
47: else if t − Δ ≤ t_last then
48:     return (true, t_last)
49: else
50:     return (false, 0)
```

each fact identified in V by calling AskRemote. AskRemote(e, id), in turn, calls the RecheckFact(id) method at entity e, which looks up the validity of the fact associated with the identifier id in e's knowledge base or in the knowledge base of the entity identified by e's $RemoteCache$ object, and then returns this status to the querier. Note that unlike its counterpart in Algorithm 4.2, RecheckFact does not remove entries from an entity's $RemoteCache$ object if the remote lookup returns a false value, as validity fluctuations are permissible under the definition of sliding window consistency; evictions from this cache will instead take place by means of

some other strategy (e.g., LRU).

After the call to RECHECKVIEW returns, CHECKSLIDING iterates over the state tuples stored in V and checks to see if each tuple was valid at the previously-saved timestamp. If this check succeeds, CHECKSLIDING indicates that V is still sliding window consistent and that all facts were verified to be simultaneously true at the aforementioned time stamp. Should this check fail but all facts were previously observed to be simultaneously valid within Δ time units of the previously-saved timestamp, this previous time of simultaneous validity is returned in support of V's continuing sliding window consistency. If neither of these conditions are true, then V is no longer sliding window consistent and CHECKSLIDING returns false. Note that the second component of the tuple returned by CHECKSLIDING indicates the last time at which the sliding window consistency condition was verified to hold; this timestamp can then be used as the t_{last} argument in future invocations of CHECKSLIDING.

We make the following claim regarding the soundness of Algorithm 4.3:

THEOREM 4.4.6. *If the series of n function calls* CHECKSLIDING$(V, \Delta, 0), \ldots,$ CHECKSLIDING(V, Δ, t^n_{last}) *made at times $t_1, \ldots, t_n$ return the values (*true*, t'_1), ..., (*true*, t'_n) then the view V is sliding window consistent on the interval $[t_1, t_n]$ using the window size Δ, provided that V was initially constructed using only* RCVRESPONSE*, was modified only through the above calls to* CHECKSLIDING*, and that $t^i_{last} = t'_{i-1}$ for all $i \geq 2$.*

Proof. By an argument similar to that used in the proofs of Theorems 4.4.1 and 4.4.3, the RCVRESPONSE function correctly constructs the initial view V. We must now show that the function RECHECKVIEW correctly updates the fact state tuples stored in V. This function utilizes the ASKREMOTE function to query the entity associated with each fact state tuple $s \in V$ regarding the associated fact's status. This triggers the RECHECKFACT function at the remote entity, which either returns a response from its local knowledge base or, if this node is an indirect node, queries the actual fact provider and returns that response to the initial querier. If the fact status returned is true and the associated validity duration overlaps an already-established validity interval for this fact even after it is adjusted to account for clock skew, then this existing validity interval is lengthened. Otherwise, if the fact is true, and the validity duration encompasses the time t at which RECHECKVIEW was invoked then the existing validity interval is replaced by the interval $[t, t]$. If neither of these cases holds, the existing validity interval is left unchanged. Since the changes made to V by RECHECKVIEW are consistent with the querier's observations *and* account for clock skew between nodes in the system, we can conclude that RECHECKVIEW

correctly updates V.

Given that RCVRESPONSE correctly creates views and RECHECKVIEW correctly updates the validity information stored in a view, we must now show that CHECKSLIDING correctly enforces the sliding window consistency condition. We proceed by induction on the number of calls made to the CHECKSLIDING function. To prove the base case, we must show that if CHECKSLIDING$(V, \Delta, 0)$ is invoked at time t and returns the tuple (true, t') then the predicate $\phi_{sliding}(V, t, t, \Delta)$ holds true. In this case, the first **if** statement in CHECKSLIDING will be bypassed, since $t_{last} = 0$, and V will be updated by a call to RECHECKVIEW. Now, CHECKSLIDING will only return true if the **if** statement at Line 43 succeeds for each $s \in V$. Since this test ensures that the validity interval for each $s \in V$ includes the time t at which CHECKINTERVAL was invoked, it implies that the predicate $\phi_{sliding}(V, t, t, \Delta)$ holds.

Assume that the predicate $\phi_{sliding}(V, t_1, t_n, \Delta)$ holds if the series of function calls CHECKSLIDING$(V, \Delta, 0), \ldots,$ CHECKSLIDING(V, Δ, t^n_{last}) is made at times $t_1, \ldots, t_n$ and returns the values (true, t'_1), ..., (true, t'_n). To complete this proof, we must now show that if CHECKSLIDING(V, Δ, t'_n) returns (true, t'_{n+1}) when invoked at time t_{n+1}, then the predicate $\phi_{sliding}(V, t_1, t_{n+1}, \Delta)$ holds. As long as the first **if** statement in CHECKSLIDING—which ensures that t_{n+1} is within Δ of the last time at which V was evaluated to be sliding window consistent—evaluates to true, then CHECKSLIDING will update the validity intervals stored in the view and return (true, t'_{n+1}). If all of the validity intervals maintained by V could be extended to include the time t_{n+1} then $t'_{n+1} = t_{n+1}$, otherwise $t'_{n+1} = t'_n$. In either case, the theorem holds. □

Although Algorithm 4.3 is shown to be sound by the above theorem, it is not ideally complete. There could very well be unobserved times between calls to the CHECKSLIDING function in which the facts comprising V are simultaneously valid. Entities can, however, balance a trade-off between proximity to ideal completeness and computational overhead by increasing the frequency of calls to CHECKSLIDING. We now prove that Algorithm 4.3 is a policy-safe modification to the Minami-Kotz distributed proof construction protocol.

THEOREM 4.4.7. *Algorithm 4.3 is a policy-safe modification to the Minami-Kotz distributed proof construction protocol.*

Proof. Algorithm 4.3 does not affect the construction of distributed proofs, so all integrity policies are still respected. To show that the confidentiality policies of nodes in the system are respected by Algorithm 4.3, we note that the only difference between the recheck procedure followed by this algorithm and that followed

by Algorithm 4.2 is that this algorithm also reveals a validity duration in addition to the fact status. Since disclosing this duration to the querying entity does not reveal any more confidential information than revealing the status of a given fact, an argument identical to that used in the proof of Theorem 4.4.4 shows that Algorithm 4.3 also respects each entity's confidentiality policies. □

4.5 Evaluation

In this section, we measure the performance impact of our consistency enforcement algorithms. The environment in which we ran our tests consisted of a 25 node cluster connected with 100Mbit Ethernet. Each node has a 3.2GHz Intel Pentium D 940 dual-core processor and 2GB RAM, and runs RedHat Linux AS 4 and Sun Microsystems' Java runtime (v1.4.2). Our system has approximately 12,500 lines of Java code, of which about 600 lines represent extensions to the core implementation of the proof construction system described by Minami and Kotz [91]. We used the Java Cryptographic Extension (JCE) framework to implement RSA and Triple-DES (TDES) cryptographic operations. A 1024-bit public key whose public exponent is fixed to 65537 was used in all of our experiments and the RSA signing operation used MD5 [103] to compute the hash value for each message to be signed. On this hardware, the RSA signature and verification operations take 6.62 ms and 0.62 ms, respectively. We used Outer-CBC TDES in EDE mode [43] to perform symmetric key operations. The length of our DES keys was 192 bits, and the padding operation in TDES operations conforms to RFC 1423 [6].

During our experiments, we measured the latency of constructing distributed proof trees using two different strategies to ensure interval consistency. These experiments utilized 25 servers, each of which was run by a different principal. Each query issued during our experiments was of the form $?grant(P, R)$ where P is a principal and R is a resource. The body of each rule in each knowledge base is of the form $a_0(c_0), \ldots, a_{n-1}(c_{n-1})$, where each a_i is a predicate symbol and each c_i is a constant. Our experiments attempted to create proof trees containing up to 35 nodes. We believe that proof trees of this size are significantly larger than would be required in most applications, thus, our results should provide guidelines about the worst-case latency for a wide array of practical applications. Authorization, confidentiality, and integrity policies were generated for each of these principals automatically and in such a fashion as to ensure that valid proof trees of the appropriate size could be constructed. For each size of proof tree analyzed during these experiments, measurements were taken during the construction of ten different proof trees of varying internal structure.

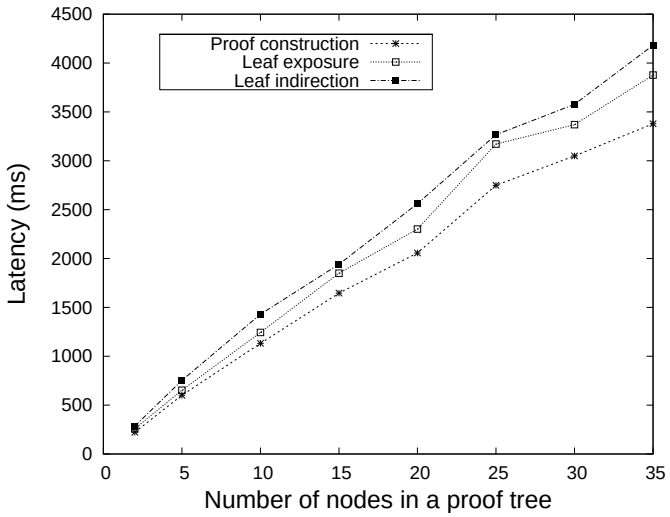

Figure 4.6: Latency for handling queries.

Figure 4.6 compares the query-handling latencies of three different proof construction algorithms; each data point is an average of 50 runs (5 runs for each of the 10 different proof trees generated per proof size). The *proof construction* curve illustrates the cost of generating the proof tree corresponding to some initial query. Note that unless Algorithm 4.1 is used, no guarantees about the consistency level of the view used to construct these proof trees can be made. The *leaf exposure* curve illustrates the cost of using the leaf exposure strategy to guarantee that proofs are generated using interval consistent views. In this case, the identities of all leaf nodes in the system are forwarded to the initial querier, who can then recheck the validity of each base fact directly. Recall from Section 4.4.3 that, in many ways, this scenario represents a time-optimal strategy for constructing interval consistent views. The *leaf indirection* curve represents the cost of ensuring that proofs are generated using interval consistent views through the use of the leaf indirection strategy described in Algorithm 4.2.

Figure 4.6 shows that these two strategies for enforcing the use of interval consistent views cost little more than generating the initial distributed proof. Specifically, the leaf exposure strategy takes only about 10–15% more time than generating the initial proof tree, while the leaf indirection strategy takes only 25–30% more time than generating the initial proof tree. These results confirm our earlier conjecture that the leaf indirection strategy is a close approximation of the time-optimal leaf exposure strategy. The leaf indirection strategy is also vastly more efficient than the naive requery strategy which, by definition, would require 100% more time than generating the initial proof tree. Although these results depend on our specific implementations of the distributed proof construction and consistency enforcement

algorithms, it is still interesting to note that it is possible to recheck proofs *much* faster than they can be constructed; this may lead to the design of more efficient distributed proof engines in the future. These efficiency results combined with Theorems 4.4.4 and 4.4.5 firmly establish the leaf indirection strategy (as implemented by Algorithm 4.2) as a low-cost, privacy-preserving method for ensuring the use of interval consistent views during the construction and evaluation of distributed proofs.

We also note that the cost of enforcing the sliding window consistency condition via Algorithm 4.3 can be calculated using the data from Figure 4.6. An execution of Algorithm 4.3 involving r rechecks of the proof requires the time indicated on the *proof construction* curve plus r times the difference between the *proof construction* and *leaf indirection* curves. As was alluded to in Section 4.4.4, this cost is significantly less than using r invocations of Algorithm 4.1, which would have a cost of r times the *proof construction* curve.

4.6 Summary

In this chapter, we explored the problem of enforcing consistency constraints on the system views used during policy evaluation in an authorization system based on distributed proof construction. In particular, we focused on enabling the use of consistent system views when evaluating policies within the proof construction framework presented by Minami and Kotz [90]. This framework complicates the view consistency problem, as the confidentiality and integrity policies declared by entities in the system may render the full details of a given proof tree unavailable to the initial querier. Further, simple signed assertions are used as facts in the system, rather than CA-issued certificates.

Within this framework, we formally defined the view consistency problem and several important levels of view consistency. We then presented efficient algorithms for enforcing three interesting levels of view consistency, proved the soundness of each algorithm, commented on the proximity of these algorithms to ideal completeness, and proved that all three algorithms represent policy-safe modifications to the underlying proof system. That is, none of these algorithms have any effect on the proper enforcement of confidentiality or integrity policies defined by entities in the system. We then quantitatively evaluated the impact of these algorithms on an implementation the proof system presented by Minami and Kotz [90]; this impact was found to be minimal. The solutions developed in this chapter generalize those from Chapter 3, which assumed that all assertions used during the proof construc-

tion process were encoded in CA-issued certificates and that each assertion used during the protocol was available to the policy evaluator for inspection.

5 Audit in the Absence of Traditional User Identities

All the world's a stage,
And all the men and women merely players;
They have their exits and entrances,
And one man in his time plays many parts

—William Shakespeare
As You Like It (II, vii)

The preceding two chapters showed how to develop mechanisms to ensure that only *safe* decisions are made by decentralized ABAC approaches to authorization when they are used in asynchronous distributed systems. The next logical question involves exactly *how* to deploy these systems. In addition to considering architectures and strategies for integrating ABAC approaches into existing legacy environments (e.g., see [78; 79]), we must also consider how the paradigm shift to ABAC systems will affect existing best practices. In particular, the lack of available identity information in attribute-based trust management systems complicates the design of audit and incident response systems, anomaly detection algorithms, collusion detection and prevention mechanisms, and reputation systems, all of which are taken for granted in traditional distributed systems.

In this chapter, we show that as two entities in an attribute-based trust management system interact, each learns one of a limited number of *virtual fingerprints* describing their communication partner. We show that these virtual fingerprints can be disclosed to other entities in the open system without divulging any attribute or absolute-identity information, thereby forming an opaque pseudo-identity that can be used as the basis for the above-mentioned types of services. To this end, we develop the Xiphos reputation system to allow reputation establishment using virtual fingerprints, rather than entities' civil identities. We discuss the trade-off between privacy and trust, examine the impacts of several attacks on the Xiphos system, and discuss the performance of Xiphos in a simulated grid computing system.[1]

[1] A portion of the material presented in this chapter was originally published as [74].

5.1 Introduction

Decentralized ABAC systems provide an effective and scalable means for making access control decisions in open systems, but depending on their deployment model, may have the side effect of virtually eliminating absolute identity information. In systems where a user's attributes are bound to a single identity certificate, this is obviously not the case. However, in more flexible systems where users may have multiple "identity" certificates or attributes represented by credentials that are not linked to their other credentials (e.g., each attribute is a separate X.509 key pair), the more traditional notion of user identity becomes blurred. This lack of absolute identity can be a double-edged sword in that it increases system scalability while also increasing user anonymity; this may not be appropriate in all application domains.

In traditional distributed computing systems, user identity forms the basis of audit and incident response systems, anomaly detection algorithms, collusion detection/prevention mechanisms, and reputation systems. As such, this functionality either does not exist or exists only in limited form in current open system proposals. In this chapter, we take a first step towards addressing this problem by describing a method for the linking and correlation of multiple identities used by the same entity in attribute-based trust management systems. We then show how these identities can be turned into *virtual fingerprints* which can be exchanged between entities in the system without leaking sensitive attribute or civil-identity information. Virtual fingerprints act much like fingerprints in the physical world in that they allow multiple actions initiated by an entity to be linked without necessarily knowing the civil-identity of their owner. The virtual fingerprints derived during a protocol execution can be exchanged between multiple users, thereby forming a solid foundation upon which the types of functionality previously described can be constructed.

In simple applications, such as audit logs or distributed black lists, virtual fingerprints can take the place of user identities in a fairly straightforward fashion. However, to more fully explore the expressive identity-matching semantics afforded by this approach, we will study the use of virtual fingerprints in a more data-intensive application. Specifically, we will show how virtual fingerprints can be used as the basis for Xiphos, a reputation system for use in conjunction with decentralized ABAC approaches. Reputation systems will be a necessary part of the open systems of the future, as current research trends are beginning to embrace distributed theorem proving approaches to access control [12; 119]. In these types of systems, proof fragments and access hints are collected from various parties in the network

and used to construct proofs of authorization. Accepting proof fragments or access hints from malicious entities could have negative consequences, including potentially unbounded searches for non-existent credentials and the risk of being denied access to a resource which one is, in fact, authorized to access. We show how virtual fingerprinting can be used as the foundation of a reputation system that will allow entities in an open system to gain confidence in information provided by others (including proof hints) without compromising each entity's desire to protect his or her sensitive credentials.

The remainder of this chapter is organized as follows. Section 5.2 describes how virtual fingerprints can be derived from the information collected during interactions in decentralized ABAC systems and discusses some target application domains for virtual fingerprinting. In Section 5.3, we describe the design of Xiphos, a reputation system in which reputations are aggregated using the virtual fingerprinting mechanism described in Section 5.2. We also discuss several deployment models for this reputation system, each of which allows for a different balance of privacy and completeness of available information. In Section 5.4, we discuss the privacy implications of the Xiphos reputation system and examine the effects of several attacks against this system. Section 5.5 presents an evaluation of our reputation system in a simulated grid computing network to demonstrate its utility and quantify its costs of deployment. We then summarize this chapter in Section 5.6.

5.2 Identity in Open Systems

Each entity, A, in an attribute-based trust management system has a finite set of credentials, $\mathcal{C}_A = \{c_1, \ldots, c_n\}$, which attest to her various attributes. Although these credentials might never explicitly reference A's civil identity (for example, they could be X.509 credentials that assert only that their owner has a given attribute), we claim that in practice, $\mathcal{C}_A$ completely describes A. In trust management systems such as PolicyMaker [22], KeyNote [21], QCM [55], Cassandra [17], and various trust negotiation proposals (e.g., [20; 68; 83; 118]), each credential is issued to exactly one owner in order to avoid the group key revocation problem. Thus, if an entity E can prove ownership of some $c \in \mathcal{C}_A$, then necessarily $E = A$.

Since entities may consider some of their credentials to be private, $\mathcal{C}_A$ is in most cases not globally available as a basis of comparison for identity establishment. However, as entities in these systems interact, they collect valuable information about one another even if no civil identity information is explicitly disclosed. Specifically, as entities A and B interact, B learns $\mathcal{D}_A^B \subseteq \mathcal{C}_A$. We will call sets such as $\mathcal{D}_A^B$

descriptions.

DEFINITION 5.2.1 (Description). *A* description *is a subset of the credentials owned by one entity which is learned by another entity in the system.*

We will use the notation $\mathcal{D}_A^B$ to represent the description of A known by B. It is important to note that for B to accept $\mathcal{D}_A^B$ as a description of A, A must demonstrate proof of ownership of each credential $c \in \mathcal{D}_A^B$ to B.[2] The collection of all such descriptions will be denoted by $\mathbb{D}$.

Over the course of multiple interactions, B can use previously obtained descriptions to recognize when he is communicating with a familiar entity. For this to be useful, however, the number of disjoint descriptions that an entity assume must be small. We assert that this is indeed the case; even though an entity can have an infinite number of self-issued or other low-value credentials, only credentials issued by *trusted* third parties will be useful in gaining access to the resources shared in an open system. It should not be possible to obtain an unlimited number of such credentials (e.g., a user should not be able to obtain two driver's licenses), which implies that the set of useful descriptions that can be assumed by any entity will necessarily be finite.

Although descriptions are useful for allowing one entity to recognize another entity with whom she has interacted previously, privacy concerns restrict descriptions from being shared between entities. This follows from the fact that entities may consider some of their attributes to be sensitive: even though B learns some credential c which belongs to A, this does not mean that any arbitrary entity in the system has the right to learn c. To allow certain information contained within a description to be shared between entities, we introduce the notion of *virtual fingerprints*.

DEFINITION 5.2.2. *The* virtual fingerprint *associated with a description* $\mathcal{D}_A^B = \{c_1, \ldots, c_k\}$ *is defined as* $\mathcal{F}_A^B = \{h(c_1), \ldots, h(c_k)\}$*, where* $h(\cdot)$ *is a cryptographic hash function and each* $h(c_i)$ *is called a* feature*. The collection of all such virtual fingerprints will be referred to as* $\mathbb{F}$*.*

The collision-resistance property of hash functions allows virtual fingerprints to be used as pseudo-identifiers in the same way as descriptions. For instance, if SHA-1 is used to derive virtual fingerprints, we expect that each person on Earth would need to hold approximately 2^{47} credentials before a collision would be found, given

[2]The only exception to this rule occurs when c is a delegated credential. In this case, $\mathcal{D}_A^B$ should contain both c and the long-term credential from which c was derived. For obvious reasons, proof of ownership of the long-term credential is not required.

that the current population is about 6.2 billion $< 2^{33}$ people. Therefore, if two virtual fingerprints overlap, their corresponding descriptions overlap, and thus the two virtual fingerprints both describe the same entity. Since virtual fingerprints mask out the details of a user's credentials, they are more likely candidates for allowing inferred pseudo-identity information to be shared between entities. It must be noted, however, that an entity may have multiple disjoint virtual fingerprints and thus even if two entities have interacted with this entity, they may not be able to agree on this fact based on virtual fingerprints alone. However, the limited number of virtual fingerprints used by an entity, A, in the system (which follows directly from the limited number of descriptions of A) implies that over time, factions of entities who know A by each of her virtual fingerprints will form. Clearly, virtual fingerprints can be used to link and correlate the actions of users in an open system without revealing their private attribute data to entities who do not know it already.

It should be noted that virtual fingerprinting cannot be used in conjunction with all types of trust management systems. For example, virtual fingerprints cannot be derived in systems that use anonymous credentials (e.g., [29; 33; 34]) or hidden credentials [60], since the credentials belonging to one entity are never fully disclosed to other entities in the system. In addition, the systems discussed in [29; 33; 34] were designed to prevent actions taken at disparate points in an open system from being linked, and thus prevent any form of distributed auditing. However, there are many types of systems that could benefit from the scalability of attribute-based trust management systems, but require the ability to audit transactions in the system so that users can be held accountable for their actions. Examples of these types of systems include grid computing systems, the semantic web, critical infrastructure management networks, joint military task forces, and disaster management coordination centers. Virtual fingerprinting can pave the way for the adoption of attribute-based trust management systems in these types of high-assurance environments by increasing user accountability and auditability. In the remainder of this chapter, we substantiate this claim by describing how virtual fingerprints can form the basis of a reputation system for use in conjunction with attribute-based access control systems such as those described in [17; 20; 21; 22; 24; 68; 83; 113; 118].

5.3 The Xiphos Reputation System

Recent research indicates that reputation systems will play an important role in the peer-to-peer and ad-hoc networks of the future (e.g., [51; 65; 85; 107]). In the context of open systems, reputation systems are of increasing importance as distributed

theorem proving approaches to access control begin to gain traction [12; 119], since accepting proof fragments or access hints from malicious entities could have undesirable consequences. However, the lack of concrete identity information in attribute-based access control systems makes designing reputation systems a difficult task.

In this section, we present Xiphos, a reputation system based on the virtual fingerprints described in Section 5.2. The reputation update equations used by Xiphos are similar to those used in other proposals and could easily be changed as better reputation update mechanisms are proposed; in fact, many of the equations presented in this section are adaptations of those presented by Liu and Issarny in [85], altered to work within our virtual fingerprint collection and analysis framework. Thus, our primary contribution is not the reputation update equations themselves, but rather the framework though which entities can record, index, and exchange virtual fingerprints obtained during their interactions in a privacy-preserving manner to formulate reputations for entities whose identities are never fully disclosed.

5.3.1 Local Information Collection

As entities in an attribute-based trust management system interact, they learn valuable information regarding one another's virtual fingerprints. Formally, as entities interact, they can store tuples of the form $T = \langle \mathcal{F} \in \mathbb{F}, r \in \mathcal{R}, \tau \in \mathbb{T} \rangle$, where $\mathcal{F}$ is a virtual fingerprint, r is a rating, and τ is the timestamp of the entity's most recent interaction with the entity described by virtual fingerprint $\mathcal{F}$. We assume that the set of all possible timestamps is $\mathbb{T}$ and that reputation ratings come from some set $\mathcal{R}$ of possible values. To simplify our discussion, in this chapter we use $\mathcal{R} = [-1, 1]$. However, in practice it will often be the case that ratings are vector quantities (i.e., $[-1, 1]^n$) that allow an entity to rate several aspects of her interaction with another entity (e.g., both the service quality and recommendation quality). All operations carried out on reputation ratings in this chapter can be carried out on vectors, so we use $n = 1$ in our formulas without loss of generality.

Over time, it is possible that some entity B will learn several non-overlapping virtual fingerprints describing another entity A. Thus, after a tuple $\langle \mathcal{F}_A^B, r, \tau \rangle$ is inserted into B's database, B must condense the set of all overlapping tuples. That is, B will remove the set of all tuples $\mathcal{T} = \{T \mid T.\mathcal{F} \cap \mathcal{F}_A^B \neq \emptyset\}$ from his database and insert a single tuple T' which is defined as follows:

$$T' = \left\langle \bigcup_{T \in \mathcal{T}} T.\mathcal{F}, \frac{\sum_{T \in \mathcal{T}} T.r \times \varphi(T.\tau)}{\sum_{T \in \mathcal{T}} \varphi(T.\tau)}, \tau_{now} \right\rangle \tag{5.1}$$

In the above equation, τ_{now} is the current timestamp and $\varphi(\cdot)$ is a function which computes a factor in the interval $[0, 1]$ that is used to scale the impact of older ratings. One possible definition of $\varphi(\cdot)$ fades ratings linearly over some duration d, though other definitions are certainly possible:

$$\varphi(t) = \begin{cases} 1 - \frac{\tau_{now} - t}{d} & \text{when } \tau_{now} - t < d, \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

Equations 5.1 and 5.2 form the basis of a local reputation system in which any entity can track her interaction history with any other entity in the absence of concrete identity information; this history can then be used as a predictor of future success. In the following subsections, we describe three ways in which entities can exchange portions of their local histories to form a system-wide reputation system.

5.3.2 A Centrally Managed Reputation System

Information Collection

The simplest types of reputation systems to reason about are systems in which a central server is responsible for storing and aggregating reputation values, such as the eBay feedback system. In a centralized deployment of Xiphos, the server will store tuples of the form $T = \langle \mathcal{F}_A \in \mathbb{F}, lc \in [0, 1], \mathcal{F}_B \in \mathbb{F}, r \in \mathcal{R}, \tau \in \mathbb{T} \rangle$ where $\mathcal{F}_A$ is a virtual fingerprint of the entity reporting the rating, lc is the server's linkability coefficient for the entity whose virtual fingerprint is $\mathcal{F}_A$, $\mathcal{F}_B$ is the virtual fingerprint of the entity being rated (as observed by the rater), r is the rating, and τ is the timestamp at which this rating was logged. Prior to discussing the calculation of reputation values based on these tuples, we must first explain how the server learns $\mathcal{F}_A$, and the mechanism through which lc is calculated.

For several reasons discussed later in this chapter, it is important that the server records one of the rater's virtual fingerprints along with each reputation rating registered in the system. One way for this to occur is for the rater to simply reveal several credentials to the server while reporting his reputation rating. Alternatively, the rater could carry out an *eager trust negotiation* [117] with the reputation server prior to submitting his reputation ratings. An eager trust negotiation begins by one party disclosing his public credentials to the other party. Subsequent rounds of the negotiation involve one party disclosing any credentials whose release policies were satisfied by the credentials that they received during previous rounds of negotiation. This process continues until neither entity can disclose more

credentials to the other.

In Xiphos, linkability coefficients are used to weight the reputation rating submitted by a particular entity based on how much the rater is willing to reveal about herself. To this end, the function $\gamma : \mathbb{D} \rightarrow [0, 1]$ is used to establish the linkability coefficient associated with a description (as defined in Section 5.2) learned about an entity. The exact definition of $\gamma(\cdot)$ will necessarily be domain-specific, but several important properties of $\gamma(\cdot)$ can be easily identified. First, low-value (e.g., self-signed) credentials should not influence the linkability coefficient associated with a description. This prevents an entity from establishing a large number of descriptions that can be used with high confidence. Second, $\gamma(\cdot)$ should be monotonic; that is, an entity should not be penalized for showing more credentials, as doing so increases the ease with which her previous interaction history can be traced. Third, to help prevent slander and self-promotion attacks, the sum of the linkability coefficients derived from any partitioning of a description should not be greater than the linkability coefficient derived from the entire description. More formally, given a description $\mathcal{D} \in \mathbb{D}$, $\forall P = \{p_1 \subseteq \mathcal{D}, \ldots, p_k \subseteq \mathcal{D}\}$ such that $\bigcap_{p \in P} p = \emptyset$, $\gamma(\mathcal{D}) \geq \sum_{p \in P} \gamma(p)$. We discuss and evaluate a particular $\gamma(\cdot)$ function which meets these criteria in Section 5.5.2.

The linkability coefficient is a good metric by which to establish a "first impression" of an entity, as a high linkability coefficient implies that an entity's previous interactions can be more easily tracked. This becomes especially meaningful if the reputation system itself stores vector quantities and can look up a "rating confidence" value for a particular user (such as the *RRep* value stored in [85]). Entities with higher linkability coefficients are more likely to have many meaningful rating confidence scores reported by other entities which could be used to weight their contributions to the system. In this chapter, we simply use the linkability coefficient as an estimate of an entity's rating confidence.

Given that the server stores tuples in the above-mentioned format, we now discuss how reputation ratings are updated. Assume that after interacting with some entity, the server determines that the tuple $T = \langle \mathcal{F}, lc, \mathcal{F}', r, \tau \rangle$ should be inserted into the database. Prior to inserting this tuple, the database first purges all prior reputation ratings reported by the entity described by $\mathcal{F}$ regarding the entity described by $\mathcal{F}'$. That is, the set of tuples $\mathcal{T}_{old} = \{T \mid (T.\mathcal{F}_A \cap \mathcal{F} \neq \emptyset) \wedge (T.\mathcal{F}_B \cap \mathcal{F}' \neq \emptyset)\}$ are deleted from the database.[3] At this point, T can be inserted. Note that user updates replace older reputation ratings rather than scaling them since users locally time-scale their own ratings according to Equation 5.1.

[3] Alternatively, these tuples could be saved for historical purposes, but marked as expired.

Query Processing

Having discussed how information is stored at the reputation server, we now describe how queries are processed. If an entity is interested in obtaining the reputation of some other entity whose virtual fingerprint is $\mathcal{F}$, he submits a query of the form $\mathcal{F}_Q \subseteq \mathcal{F}$ to the reputation server. To compute the reputation for the entity with the virtual fingerprint $\mathcal{F}_Q$, the server must first select the set of relevant tuples $\mathcal{T}_Q = \{T \mid T.\mathcal{F}_B \cap \mathcal{F}_Q \neq \emptyset\}$. If any subset $\mathcal{T}_Q^A$ of the tuples in $\mathcal{T}_Q$ have overlapping $\mathcal{F}_A$ components, these tuples will be removed from $\mathcal{T}_Q$ and replaced with a summary tuple of the form:

$$\left\langle \bigcup_{T \in \mathcal{T}_Q^A} T.\mathcal{F}_A, max(\{T.lc \mid T \in \mathcal{T}_Q^A\}), \bigcup_{T \in \mathcal{T}_Q^A} T.\mathcal{F}_B, \frac{\sum_{T \in \mathcal{T}_Q^A} T.r \times \varphi(T.\tau)}{\sum_{T \in \mathcal{T}_Q^A} \varphi(T.\tau)}, \tau_{now} \right\rangle \tag{5.3}$$

This duplicate elimination prevents the server from overcounting the rating of a single entity A who knows the subject of the query by more than one disjoint virtual fingerprint, each of which overlaps $\mathcal{F}_Q$. Let $\mathcal{T}'_Q$ denote the results of performing this duplicate elimination process on $\mathcal{T}_Q$. Given $\mathcal{T}'_Q$, the reputation associated with the query $\mathcal{F}_Q$ is defined by the following equation:

$$r_Q = \frac{\sum_{T \in \mathcal{T}'_Q} (T.lc \times \varphi(T.\tau) \times T.r)}{\sum_{T \in \mathcal{T}'_Q} (T.lc \times \varphi(T.\tau))} \tag{5.4}$$

In short, the reputation returned by the server is the weighted average reputation rating of entities matching the virtual fingerprint $\mathcal{F}_Q$, where each reputation rating is weighted based on both the linkability coefficient of the rater (which acts as an estimator of her rating confidence value) and the age of the reputation rating. An interesting area for future work involves exploring the applicability of various weighting schemes adapted from the information retrieval field. For example, in some situations, it may be advantageous to weight each tuple based on its degree of similarity to the query. In this case, similarity could be measured either in terms of the number or value of overlapping features that a tuple's $\mathcal{F}_B$ component shares with $\mathcal{F}_Q$.

The curious reader might wonder why the set intersection operator is used to define $\mathcal{T}_Q = \{T_i \mid T_i.\mathcal{F}_B \cap \mathcal{F}_Q \neq \emptyset\}$ as the set of matching tuples for a query $\mathcal{F}_Q$ rather than the transitive closure of this operator. While in a network with only honest participants, the transitive closure would give more accurate reputation ratings, it

would cause incorrect results to be calculated if cheaters are present in the system. As an illustration, consider a system in which some entity E (with virtual fingerprint $\mathcal{F}_E$) is known to have an excellent reputation. A malicious entity M (with virtual fingerprint $\mathcal{F}_M$) could then inflate his reputation by having some third party N (with virtual fingerprint $\mathcal{F}_N$) report a rating for the "entity" whose virtual fingerprint is $\mathcal{F}_E \cup \mathcal{F}_M$, thereby causing the tuple $T = \langle \mathcal{F}_N, lc_N, \mathcal{F}_E \cup \mathcal{F}_M, r, \tau \rangle$ to be inserted into the central database. If the transitive closure of the set intersection operation was then used to define $\mathcal{T}_Q$, any searches for M's reputation would then also include all ratings for E, thereby inflating M's reputation. For this reason, we use only set intersection for query matching, as entities can submit queries derived from virtual fingerprints which they have *verified* to belong to another entity. This further justifies the use of the linkability coefficient as a first impression of another entity, since as the linkability coefficient increases towards 1.0, the information included in $\mathcal{T}_Q$ approaches completeness.

5.3.3 A Fully Distributed Reputation System

We now describe a fully distributed deployment of Xiphos. In this model, entities calculate reputation ratings for other entities by querying some subset of the other entities in the system and aggregating the results from their local databases. As in the centralized model, queries are of the form $\mathcal{F}_Q \in \mathbb{F}$. Each node queried selects from their local database all tuples which overlap $\mathcal{F}_Q$ (i.e., $\mathcal{T} = \{T \mid T.\mathcal{F} \cap \mathcal{F}_Q \neq \emptyset\}$) and then creates a summary tuple of the form $T = \langle r_Q, \tau \rangle$ to return to the querier. If only a single tuple T' matches the query, then its r and τ components are used to form T; otherwise, Equation 5.1 is used to generate a tuple whose r and τ components are used.

Upon receiving each of these summary tuples, the querier then augments them by adding the linkability coefficient that she has associated with the entity which sent the result. This linkability coefficient can either be cached from a previous interaction, the result of an eager trust negotiation initiated by the querier, or calculated from a set of credentials sent by the other entity along with the summary tuple. Given this collection of augmented summary tuples, $\mathcal{T}_Q$, the querier then computes the reputation rating of the entity whose virtual fingerprint is characterized by $\mathcal{F}_Q$ as follows:

$$r_Q = \omega_{local} \times r_Q^{local} + (1 - \omega_{local}) \times \frac{\sum_{T \in \mathcal{T}_Q} (T.lc \times \varphi(T.\tau) \times T.r)}{\sum_{T \in \mathcal{T}_Q} (T.lc \times \varphi(T.\tau))} \tag{5.5}$$

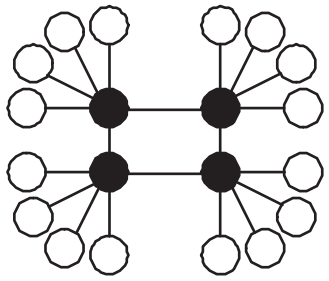

Figure 5.1: A simple super-peer network (super nodes shown in black).

The term $\omega_{local} \in [0, 1]$ represents a weighting factor which allows the querier to determine how much of the reputation rating that she calculates should be based on her previous interactions with the subject of a query (denoted by r_Q^{local}) versus the reputation ratings reported by other entities in the system. For instance, using $\omega_{local} = 0$ would mean that the reputation ratings provided by other entities will be used exclusively and any local reputation score will be ignored. In addition to choosing the weight given to the reputations returned by others, users must manually balance the time they spend querying other nodes with the accuracy of the reputation rating that they hope to derive.

5.3.4 A Reputation System for Super-Peer Network Topologies

The final deployment model that we consider is a reputation system built on top of a super-peer network. Super-peer networks [120] are peer-to-peer networks which leverage the heterogeneity of nodes in the network by using nodes with higher bandwidths and faster processors to act as intelligent routers which form the backbone of the network. In these networks, a small number of so-called "super nodes" act as gateways for a large number of standard peers. Figure 5.1 shows a simple super-peer network topology.

In this model, each super node is assumed to have complete information regarding the virtual fingerprint to reputation bindings stored by each of its client peers; that is, each super node acts as a centralized server as described in Section 5.3.2. Given a query $\mathcal{F}_Q$, a super node then uses Equations 5.3 and 5.4 to compute a local reputation rating, r_Q^S, based on the ratings provided by its client peers. However, in addition to calculating this local reputation rating, the super node can also include the reputations reported by other super nodes. After reissuing the query to each other super node and obtaining $\mathcal{T}_Q$, the set of resulting summary tuples calculated using Equations 5.3 and 5.4, the super node computes the aggregate reputation in response to the query $\mathcal{F}_Q$ as follows:

$$r_Q = \omega_S \times r_Q^S + (1 - \omega_S) \times \frac{\sum_{T \in \mathcal{T}_Q} (T.lc \times \varphi(T.\tau) \times T.r)}{\sum_{T \in \mathcal{T}_Q} (T.lc \times \varphi(T.\tau))} \tag{5.6}$$

As in the fully distributed model, ω_S is a weighting factor which determines how much the reputation rating calculated from the super node's local peer group is weighted in comparison to the reputation ratings returned by all of the other super nodes.

5.4 Discussion

In this section, we discuss the privacy concerns associated with each deployment model of the Xiphos system. We see that Xiphos is in fact a double-edged sword, and that system architects must make explicit choices regarding balancing privacy preservation and completeness of available information. We then discuss several well-known attacks on reputation systems and describe their effects on Xiphos.

5.4.1 Privacy Considerations

Though reputation systems will form a necessary part of the open systems of the future, it is important to note that the information that they provide comes at a cost. In particular, there is a very clear trade-off between preservation of user privacy and the completeness of information obtained through the reputation system. We now identify the threats to user privacy which manifest themselves in each of the deployment models presented in Section 5.3.

Possible Privacy Violations

We have identified three types of potential privacy violations that may occur as a result of the Xiphos system: leakage of interaction history, discovery of groups of entities with similar attributes, and inference of particular attribute information. Interaction history leaks occur in the centralized and super-peer deployments of the Xiphos system any time that one entity registers a reputation rating for another. This action allows the super peer or central server to infer that the rater and the ratee have interacted in the past. In the fully distributed deployment model, any time that A answers a query issued by B, B can infer that A has interacted with the subject of his query. However, leakage of interaction history occurs in every other reputation system that we are aware of, thus we do not discuss it further here.

The second type of privacy violation occurs as a central server or super peer collects large numbers of reputation tuples. Recall that these tuples are of the form $T = \langle \mathcal{F}_A, lc, \mathcal{F}_B, r, \tau \rangle$. After building a substantial database, a malicious server can select all tuples whose $\mathcal{F}_B$ component overlaps a given $\mathcal{F}_Q$ exactly. We now claim that the $\mathcal{F}_A$ components of these matching tuples determine a set of entities in the server's view of the open system who have similar attributes. The justification of this claim comes from the fact that each entity described by some $T_i.\mathcal{F}_A$ was able to determine the same virtual fingerprint for the entity matching $\mathcal{F}_Q$. Thus, each of these entities was able to unlock each of the credentials used to derive $\mathcal{F}_Q$, a feat which requires that each of these entities be able to satisfy the same set of credential release policies. Because these release policies may include disjunctions, we cannot determine that each matching $T_i.\mathcal{F}_A$ has the *same* set of defining attributes, though we can claim that these entities are *similar* in some respects. The similarity of these entities is directly correlated with the restrictiveness of the release policies protecting the credentials used to derive $\mathcal{F}_Q$; more restrictive policies lead to more closely related entities.

The third type of privacy violation allows certain entities in the system to infer attributes possessed by another entity in the system. In the centralized and super-peer models, this attack is an extension of the previously discussed attack. Consider the case where a server S knows the description $\mathcal{D}_A^S$ of a node A. Let us also assume that some $c \in \mathcal{D}_A^S$ is protected by a release policy, p, which is also known to S (e.g., as a result of some previous interaction). S can then form a query $\mathcal{F}_Q = \{h(c)\}$ and process it using the technique described above, thereby learning the virtual fingerprints of a group of entities who can satisfy p. Since S knows p, he then knows not only that each entity that matched his query is related *somehow*, but also that they satisfy p; that is, S can infer the attributes which cause the similarities between the nodes which match his query.

A Balancing Act

To an extent, choosing an appropriate deployment model for the Xiphos system can mitigate these attacks. The centralized model makes these attacks easier to carry out, as the server has complete information regarding the reputation tuples registered with the system. By using a super-peer deployment, the information flow is restricted greatly. Both the group discovery and attribute inference attacks are limited to occurring within a single peer group, since super nodes do not have access to each others' databases. Thus, if client nodes restrict their information sharing to super nodes whom they can trust (e.g., super nodes with Better Business Bureau

memberships or TRUSTe-issued privacy policies), then they can have some assurance that the super node will not abuse their partial information to carry out these attacks. Limiting the size of peer groups managed by each super node further restricts these attacks. It should also be noted that using the super-peer deployment model does not sacrifice the completeness of information available, as ratings registered by every peer are still included as the contribution of each super node is folded into the reputation rating calculated using Equation 5.6. However, unless each super node has a roughly equivalent number of members, ratings may be biased towards the opinions of entities at super nodes with fewer members. Additionally, unless super nodes coordinate to ensure that there is no overlap between their respective peer groups, the accuracy of the reputation ratings calculated using this method may suffer, as malicious peers could register ratings at multiple super nodes.

These attacks can be further limited by using the fully distributed deployment model, as no entity in the system has any sort of complete information. Each entity is restricted to querying a limited number of other entities in the system, as querying each node in turn becomes inefficient as the size of the network grows. Additionally, when issuing the query $\mathcal{F}_Q$, an entity A cannot be sure if the responding entities have matched all of $\mathcal{F}_Q$ or simply some $\mathcal{F}' \subset \mathcal{F}_Q$. This implies that A must carry out the group discovery or attribute inference attacks by issuing queries $\mathcal{F}_Q$ where $|\mathcal{F}_Q| = 1$ to ensure that all matches returned are total matches. Note also, that A will most likely need to know c where $\mathcal{F}_Q = \{h(c)\}$, as otherwise she is simply guessing that $\mathcal{F}_Q$ is an "interesting" virtual fingerprint, which may often be a difficult task. This implies that A is very likely to know p, the release policy for c, as she satisfied p to learn c in the first place. In this respect, the group discovery attack is eliminated, as A is forced to carry out the stronger attribute inference attack. The attribute inference attack is itself no more feasible than trying to determine whether the attribute a attested to by c is possessed by each node in the network directly (e.g., by means of an eager negotiation or another resource access request protocol), thus this attack is no more feasible with Xiphos in place than it would have been without it. This implies that attacks which cause the aforementioned privacy violations can be virtually eliminated by using the fully distributed deployment model, though at the cost of losing the completeness of reputation information.

In addition to leveraging the privacy versus completeness trade-off which exists in the Xiphos system, another possible avenue for the prevention of privacy-related attacks involves the use of *obligations*. Obligations are requirements that can be attached to personal information in certain types of trust management systems.

For instance, the owner of a digital medical record might attach an obligation to that record requiring that her health care provider send her an email any time this record is shared (e.g., while filing a referral to another physician). In these types of systems, it would be possible for entities to attach obligations to their credentials which limit the ways that other entities can disclose virtual fingerprints including hashes of these credentials. For example, an entity could indicate that any virtual fingerprint including a hash of her Department of Energy security clearance credential may only be released to a reputation server operated by the U.S. government. These types of obligations allow users to reap the benefits of the Xiphos reputation system while still maintaining some control over their private information. We expect that most entities will allow at least some "interesting" subset of their credential hashes to be included in virtual fingerprints because they will likely interact with other entities who require the ability to obtain their reputation rating prior to interaction. Note that in most systems obligations are not guaranteed to be enforced; that is, obligations more closely resemble preferences rather than demands and thus a malicious entity could still leak "unauthorized" virtual fingerprints to reputation services. However, a malicious entity could also post the *actual credentials* associated with these virtual fingerprints in an open forum, so the threat of leaked virtual fingerprints is minimal.

5.4.2 Attacks and Defenses

There are several types of well-known attacks that can be launched against reputation systems in hopes of biasing the reputations reported by the system. Specifically, Hoffman et al. identify five classes of attacks against reputation systems: self-promoting, whitewashing, slandering, orchestrated, and denial of service attacks [59]. In this section, we address the whitewashing, slandering, self-promoting, and orchestrated classes of attack and discuss their effects on the Xiphos system. We also discuss two potential attacks on the Xiphos system itself. We do not discuss denial of service attacks, as they do not typically target the calculation mechanism of the reputation system, which was the primary focus of this chapter.

Whitewashing. One common attack against reputation systems is whitewashing, which occurs when a user sheds a bad reputation by establishing a new identity in the system. In some systems, this is as simple as reconnecting to the network to obtain a new node identifier, while in others it may involve establishing a new pseudonym (e.g., email address) by which one is known to the system. In Xiphos, reputation ratings are associated with virtual fingerprints. As discussed in Sec-

tion 5.2, each user has only a limited number of virtual fingerprints, which are uniquely determined by the set of credentials that she possesses. Obtaining new identities thus reduces to establishing new virtual fingerprints, which requires that a user obtain *all* new credentials, as *any* overlap will link this entity to old ratings in the system. If users are routinely required to utilize multiple credentials, this process is likely to be time consuming and involve multiple certificate authorities, thereby making whitewashing a very impractical attack for *habitual* cheaters.

Slander and Self-Promotion. In reputation systems that either do not track the identity used to register a rating or allow for easily obtaining multiple identities, it is possible for an entity to register multiple ratings for a single entity and thus have their opinion overcounted. In Xiphos, the virtual fingerprinting system can be used to limit the number of claims that an entity can register with the system. Entities have only a finite number of disjoint virtual fingerprints which can be used to register claims and thus can only register a finite number of reputation ratings for other entities in the system. In addition to capping the number of ratings that an entity can register, the virtual fingerprint system also limits the benefits of registering multiple ratings. A properly designed $\gamma(\cdot)$ function will assign lower linkability coefficients to ratings associated with a small rater virtual fingerprint than it will to ratings associated with large rater virtual fingerprints. This means that given a properly designed $\gamma(\cdot)$ function, an entity's influence on the overall rating of another entity will be less if she registers multiple ratings using a large number of small virtual fingerprints than it would have been if she had registered only a single rating using the union of each smaller virtual fingerprint. Such a $\gamma(\cdot)$ function is discussed in Section 5.5.2.

Orchestrated. In an orchestrated attack, a colluding group of attackers leverages multiple attack strategies in an attempt to subvert the system. For example, one group of nodes might misbehave while another group of nodes attempts to artificially inflate the ratings of the misbehaving nodes. While Xiphos cannot prevent participants in the system from colluding, the ability of virtual fingerprints to limit the influence of individual misbehaving nodes (as discussed above) does limit the effectiveness of this class of attacks.

Exploiting $\varphi(\cdot)$. One attack against Xiphos itself involves exploiting the use of the $\varphi(\cdot)$ function. Recall that $\varphi(\cdot)$ is used to weight the contribution of a single tuple to the overall reputation calculated for a query. In the centralized and super-peer deployment models, entities in the system may try to increase their influence

by repeatedly updating their ratings for other entities in the system to keep them current. In the absence of certified transactions and synchronized clocks, there is little that can be done to prevent this problem. However, this attack will likely have little influence on the ratings calculated by the central server if the majority of the users in the system remain honest. Nonetheless, investigating mechanisms for providing general-purpose certified transaction support is an important area of future work.

Opinion Erasure. One last attack on which we comment occurs when a malicious party M is able to steal some set of credentials $\mathcal{C}'_A \subseteq \mathcal{C}_A$ from another entity A. If M then submits a reputation rating for some entity B described by the virtual fingerprint $\mathcal{F}_B$ while posing as A (by using the stolen credentials $\mathcal{C}'_A$), this rating will overwrite the rating previously stored by A. For this attack to be successful, A must have previously rated B. Though this attack is serious, it is possible in any system in which one entity is able to effectively steal the identity of another (e.g., by guessing another entity's password). Due to the fact that users in attribute-based trust management systems have many identities (which we have referred to as *descriptions* in this chapter), the design of usable and secure identity management systems is an important research challenge.

5.5 Evaluation

In this section, we present a simulation study conducted to evaluate the performance and utility of the Xiphos reputation system.

5.5.1 Experimental Setup

Simulating an attribute-based trust management system presents several interesting challenges, including modeling the distribution of credentials throughout the system and determining the assignment of release policies to the credentials held by entities in the system. To overcome these difficulties, we chose to simulate a constrained grid computing system rather than a general-purpose open system. Figure 5.2 illustrates the credential ontology used in our simulated network.

In this network, we assume that there are two types of entities: users and resources. Users represent humans interested in using the computing grid to carry out some task, while resources represent things such as computing clusters, mass storage

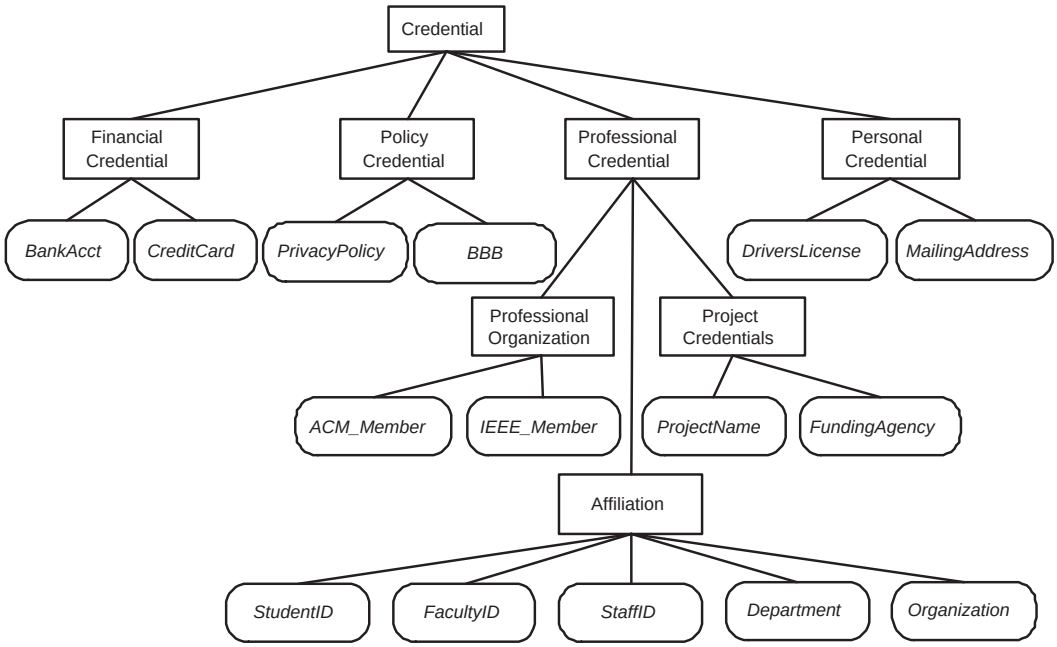

Figure 5.2: The credential ontology used in the evaluation scenario.

devices, wave tanks, and visualization facilities. Our experiments analyzed the interactions that took place in networks of various sizes generated as follows. For a network consisting of N hosts, we assume that $0.8N$ of these hosts are users, while the remaining $0.2N$ of the hosts are resources. Users and resources are randomly generated and assigned credentials and credential release policies in accordance with Table 5.1. The number of credentials of each type that are assigned to a given user or resource is chosen uniformly at random from the quantities listed in the "Users" and "Resources" columns of this table, respectively; in situations where multiple release policies are indicated for a single credential type, one is chosen uniformly at random for each credential generated. Computing grid resources are assigned access policies according to Table 5.2. The collections of hosts and resources are considered to be disjoint and are assumed to be ordered in decreasing order of popularity. We assume that there are no ties with respect to the popularity of nodes in the system. We assume that users randomly interact with both other users and with resources. Resources passively accept incoming interactions (e.g., job submissions) and do not initiate any interactions of their own.

We then simulated the interactions that would occur in this network over the course of multiple days. Each day, every user interacts with between 10 and 30 randomly-chosen entities in the network. 80% of these interactions are with other users in the network, while the remaining 20% are with resources. These interactions are chosen such that the number of incoming connections is distributed over the collec-

Credential Type (Abbrev.)	Users	Resources	Release Policy
Professional Organization (po)	0–2		*none*
Organization (o)	0–1	1	*none*, pp
Department (d)	1–2	1	*none*, pp, po
ProjectName (pn)	1–4	0–4	*none*, fa = F, bbb, pp
FundingAgency (fa)	1–2	0–2	*none*, bbb, pp
BankAcct (ba)	0–1	0	pp $\vee$ bbb, pp $\wedge$ bbb
CreditCard (cc)	0–3	0	pp $\vee$ bbb, pp $\wedge$ bbb
DriversLicense (dl)	0–1	0	*none*, pp
MailingAddress (ma)	0–2	0	*none*, pp
StudentID (s), *FacultyID* (f), or *StaffID* (st)	1	0	*none*
PrivacyPolicy (pp)	0	0–1	*none*
BBB (bbb)	0	0–1	*none*

Table 5.1: Credential distribution used in the evaluation scenario. The variable F represents a particular funding agency.

Type	Description	Policy
1	Project specific	((d = 'CS') $\vee$ (d = 'ECE')) $\wedge$ p $\in \{P_1, \ldots, P_n\}$
2	Funding agency	((d = 'CS') $\vee$ (d = 'ECE')) $\wedge$ fa = F
3	Academic	((d = 'CS') $\vee$ (d = 'ECE')) $\wedge$ (s $\vee$ f)
4	Paid academic	(s $\vee$ f) $\wedge$ (ba $\vee$ cc)

Table 5.2: Four types of resource access policies. The variables P_1–P_n represent specific projects and F represents a specific funding agency.

tions of users and resources according to Zipf distributions [124]. As nodes interact, they obtain virtual fingerprint information about one another and the initiating node registers both local and centralized ratings for their satisfaction with the interaction as described in Section 5.3.

As mentioned in Section 5.3, the equations used to determine reputation in Xiphos are very similar to those used in more traditional reputation systems. As such, our experiments do not simulate the convergence of these equations as this process has been simulated elsewhere in the literature. Specifically, our system uses reputation update equations similar to those whose convergence behavior was studied in [85]. In the remainder of this section, we focus on measurements of utility that are specific to the Xiphos system. Namely, we explore a particular $\gamma(\cdot)$ function designed for our grid computing scenario, examine the storage requirements for nodes participating in the Xiphos system, and examine query execution time as a function of database size.

5.5.2 The Effects of $\gamma(\cdot)$

It has long been observed that the concept of trustworthiness used in both physical and virtual interactions is heavily context-bound [86]. For instance, most people would be more likely to accept tax advice from an accountant rather than a hair stylist. We can leverage this notion of context sensitivity to simplify the task of defining the $\gamma(\cdot)$ function for a given environment. In some sense, the ontology presented in Figure 5.2 quantifies the exact context *relevant* to assessing the trustworthiness of entities in our grid computing scenario; this is very similar to the use of controlled vocabularies in information retrieval. While entities in the system are very likely to have numerous other credentials and attributes, the relevance of these credentials to establishing the user's trustworthiness in the *context* of grid computing is likely to be minimal. This limited contextual scope leads to a simple definition of $\gamma(\cdot)$ that meets the requirements identified in Section 5.3.2.

According to Table 5.1, users can have at most 19 credentials described by the ontology shown in Figure 5.2; resources can have at most 10 credentials described by this ontology. Note also that only resources will have *BBB* or *PrivacyPolicy* credentials. From this information, we can derive one possible instantiation of the $\gamma(\cdot)$ function:

$$\gamma(\mathcal{D}) = \begin{cases} \frac{|\mathcal{D}|}{10} & \text{if } \mathcal{D} \text{ contains a } \textit{PrivacyPolicy} \text{ or } \textit{BBB} \text{ credential,} \\ \frac{|\mathcal{D}|}{19} & \text{otherwise.} \end{cases} \tag{5.7}$$

This function assigns a linkability coefficient to a description consisting of credentials from the ontology shown in Figure 5.2 by comparing the number of exposed credentials to the maximum number of possible credentials that could have been included. Note that any credentials outside of this ontology are explicitly ignored because they are considered to be out of context. This definition of $\gamma(\cdot)$ clearly satisfies the criteria described in Section 5.3.2 and has the added advantage of encouraging resources to disclose their *BBB* and *PrivacyPolicy* credentials, as this identifies them as resources and assigns more weight to the credentials that they do show. Note that in many cases, this definition of $\gamma(\cdot)$ will assign relatively low linkability coefficients to entities, as few entities are likely to have the maximum number of possible credentials. However, Xiphos uses $\gamma(\cdot)$ only as a *relative* weighting function so this definition is satisfactory.

Figure 5.3 shows the average linkability coefficient assigned to entities in networks of 10,000–100,000 users; each data point represents the average over 10 randomly generated networks. We see that the average linkability coefficient varies slightly

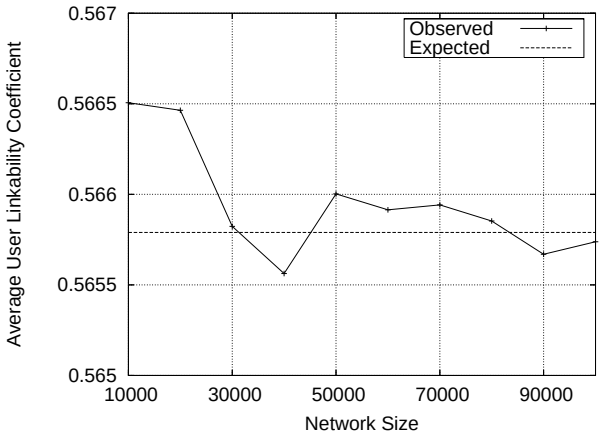

Figure 5.3: Average linkability coefficients for entities with credentials allocated according to Table 5.1.

around the expected value of 0.5658. This implies that an all-powerful attacker (i.e., an attacker with the maximum number of credentials which will be weighted by $\gamma(\cdot)$), has no more than 1.77 times the influence of an average user on the system. This assumes that the attacker cannot convince certificate authorities to issue him duplicate credentials (e.g., two driver's licenses). Therefore, if the attacker wanted to have both a "good" virtual fingerprint and a "malicious" virtual fingerprint (which must obviously be disjoint), at least one of these will have a below-average linkability coefficient, and thus less influence on the reputation scores calculated by Xiphos; average attackers are affected to an even greater degree. This shows that the linkability coefficient is useful not only for developing a "first impression" of entities in the system, but also for preventing certain types of attacks.

As reputation systems begin to be used in systems with wider contexts (e.g., the semantic web), it is important that the reputations calculated account for this context as well [85]. If Xiphos deployments wish to consider the context of the system (the possibility of which was alluded to in Section 5.3.1), the $\gamma(\cdot)$ function used should be implemented as a family of functions with one relevant member for each context considered. Other interesting future work in this area involves exploring non-uniform weighting schemes for the credentials considered by $\gamma(\cdot)$. This will allow credentials of various relevance to impact the linkability value of a particular description differently.

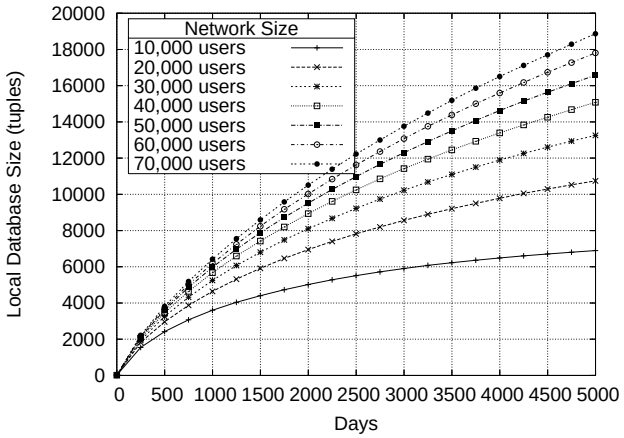

Figure 5.4: Growth of local reputation databases over time for networks ranging in size from 10,000–70,000 entities.

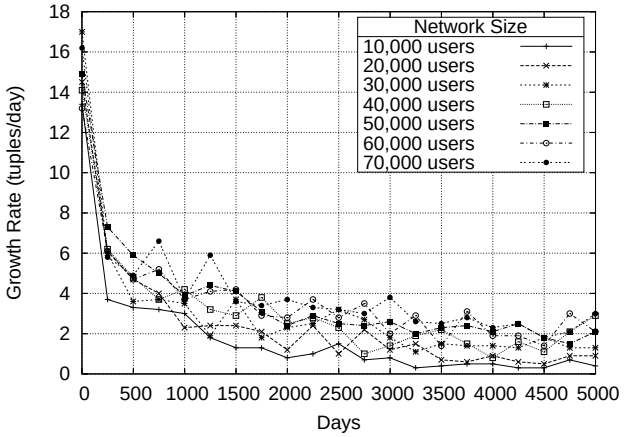

Figure 5.5: Daily change in size of local databases over time for networks ranging in size from 10,000–70,000 entities.

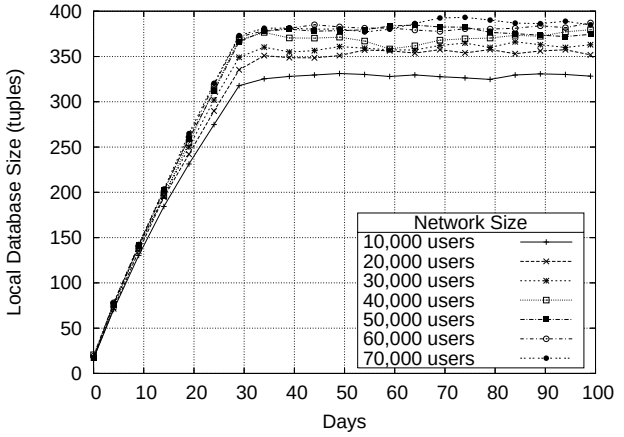

Figure 5.6: Growth of local reputation databases over time for networks ranging in size from 10,000–70,000 entities, when tuples over one month old are evicted daily.

5.5.3 Database Growth

Local Database Growth

As users in our simulations interact with resources and other users in the system, they update their local databases as defined in Section 5.3.1. This implies that over time, a host's local database will continue to grow in size and could include up to NF entries, where N is the size of the network and F is the average number of virtual fingerprints by which the host knows each entity in the network. In our simulations, we assumed that the participants in the network were honest and thus $F \approx 1$. This is not an unrealistic assumption, as if a host is known by many virtual fingerprints, the linkability coefficients associated with each virtual fingerprint and thus her overall reputation rating will be low and thus unlikely to remain in a given host's local database for very long.

For each network size simulated, we created 10 random networks and calculated the average growth of local databases in these networks over the course of 5000 days. Figure 5.4 shows this average growth assuming that nodes had unlimited storage and did not evict infrequently used tuples. This is an upper bound on tuple storage, as it effectively sets $d \geq 5,000$ days in Equation 5.2. For a network of size 10,000 (a large grid computing network by today's standards), we see that the average database size is less than 7,000 tuples after 5000 days of execution. Figure 5.5 shows the average daily growth of a local database over the same period of time. These databases grow rapidly at first but then taper off over time. Due to the long tail of the Zipf distribution, it is unlikely that this daily growth will reach zero within the lifetime of any deployed system.

Instead of requiring that each host maintain their complete interaction history, we can allow them to discard tuples that are more than one month old (effectively simulating the effect of using $d = 30$ days in Equation 5.2); this reduces storage requirements drastically. Figure 5.6 shows that storage for networks of all sizes tends to quickly stabilize at between 325 and 400 tuples, far less than the 6,000 to 19,000 tuples shown in Figure 5.4. The average daily change in local database size stabilizes around zero, as shown in Figure 5.7.

This decrease in local database size comes at the cost of forgetting about previous interactions that had unfavorable outcomes. Figure 5.8 shows the average growth of local databases when old tuples with favorable results were evicted from the database after they were 30 days old but unfavorable results were kept indefinitely. We assumed that an evenly distributed 20% of the nodes in the network were bad. This policy allows nodes to learn from history by keeping their bad memories while

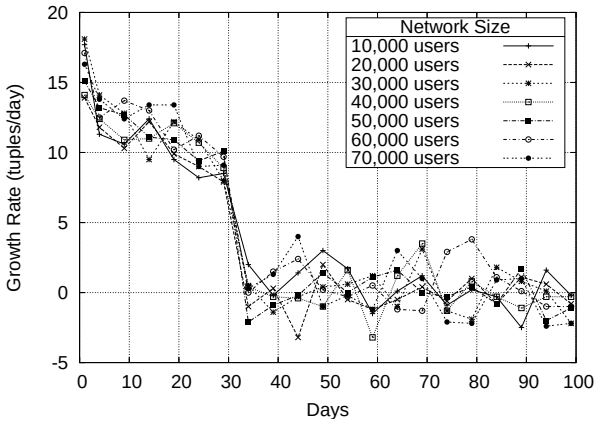

Figure 5.7: Daily change in size of local databases over time for networks ranging in size from 10,000–70,000 entities, when tuples over one month old are evicted daily.

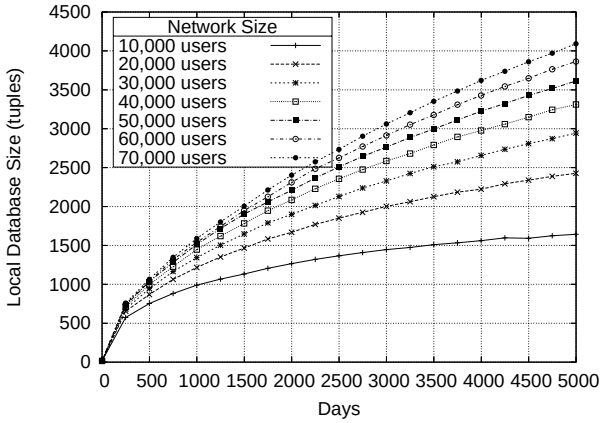

Figure 5.8: Growth of local reputation databases over time for networks ranging in size from 10,000–70,000 entities, when tuples for good interactions over one month old are evicted daily.

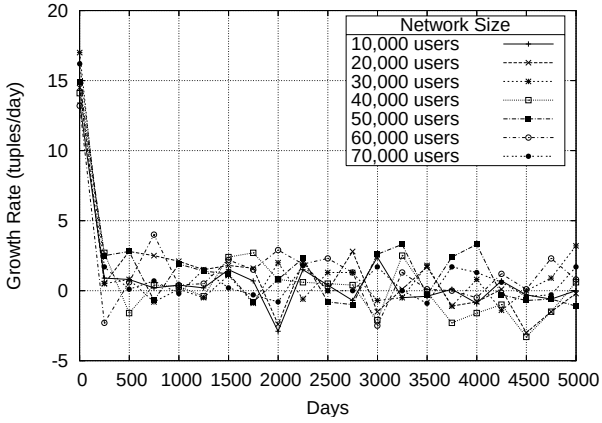

Figure 5.9: Daily change in size of local databases over time for networks ranging in size from 10,000–70,000 entities when tuples for good interactions over one month old are evicted daily.

reclaiming space by purging obsolete favorable memories. Note the slower growth rate when compared to Figure 5.4. Figure 5.9 shows the average daily change in local database size in this scenario. For this type of strategy to be effective, however, the definition of $\varphi(\cdot)$ presented in Equation 5.2 would need to be modified. Given these favorable results for very simple eviction policies, exploring more complicated eviction policies could prove to be a fruitful area of future work.

Central Database Growth

The database stored by a central server or super node is necessarily larger and more complex than those stored by other nodes in the system. In fact, a naive implementation of a central server would need to store NA tuples, where N is the number of entities in the system who report ratings to this central server and A is the average size of each entity's local database. Upon examining Figures 5.4, 5.6, and 5.8, we see that this database would quickly become enormous! In order to keep query execution times reasonable, it is clear that optimizations must be made at these central points.

The database size itself is not likely to be a problem for centralized Xiphos servers; rather, the time needed to process queries on exceedingly large databases will be the bottleneck. To address this, centralized Xiphos servers could allow interested users to become *members* of their service. Members first register with Xiphos by exposing some number of public credentials. At this point, the server creates a member entry in its database for this entity; member entries are of the form $\langle \mathcal{F} \in \mathbb{F}, n \in \mathbb{R}, d \in \mathbb{R} \rangle$, where $\mathcal{F}$ is the virtual fingerprint derived from the credentials exposed by the entity. The server then precomputes a partial reputation rating for $\mathcal{F}$ by using Equations 5.3 and 5.4 on the entire database (containing $O(NA)$ tuples). To do this, the numerator of Equation 5.4 is stored in the n field of the member entry and the denominator of Equation 5.4 in the d field of the same tuple. These precomputed reputation ratings will be refreshed on a time-available basis by the Xiphos server and thus will not reflect the exact reputation rating for a given user, but rather will act as an estimator for that value.

Processing a query $\mathcal{F}_Q$ then involves selecting all member entry tuples which overlap $\mathcal{F}_Q$ and combining their corresponding partial reputations. More formally, if $\mathcal{T}_Q$ is the set of all member entries whose $\mathcal{F}$ component overlaps $\mathcal{F}_Q$, then the final reputation estimate is calculated as follows:

$$\widehat{r_Q} = \frac{\sum_{T \in \mathcal{T}_Q} T.n}{\sum_{T \in \mathcal{T}_Q} T.d} \tag{5.8}$$

There are at most NF member tuples, where N is the number of entities recording reputation ratings at this server and F is the average number of distinct virtual fingerprints used by each entity. As we will see in Section 5.5.2, in our grid computing scenario $F \ll A$, meaning that the use of member entries will greatly reduce the number of tuples that must be accessed to answer queries. However, this reduction comes at the cost of introducing overcounting in the event that the same entity reports reputation ratings for multiple member entries, all of which match a given query. Further investigation will be required to fully evaluate the utility of this type of tuple-reduction method.

5.5.4 Query Execution Time

Now that we see how the size of each local database grows over time, we examine the average time required to process queries as a function of database size. To this end, we have implemented a prototype of the client portion of the Xiphos system in the Java programming language. Our implementation searches local databases in a linear fashion (i.e., the stored records are not indexed), making it a lower bound on the performance that one would expect in practice. The query execution times reported are averages over 1000 queries submitted to 10 randomly populated local databases. These queries were run on an IBM T40p laptop with a 1.6GHz Pentium M processor and 1 GB of memory running Windows XP. We consider this machine as a lower bound of what a scientist would use to submit and track jobs on a computational grid. Clearly, resources in the system would be much more powerful than this.

Figure 5.10 shows the execution time (in milliseconds) for queries submitted to databases ranging in size from 0 to 50,000 tuples. The linear trend is not surprising, as we implemented the $O(N)$ algorithm that follows directly from the description in Section 5.3.3. Figure 5.11 shows the number of queries per second that can be processed for local databases in the 10,000–50,000 tuple size range. For the database sizes shown in Figure 5.8, we feel that the query throughput afforded by even our prototype implementation of Xiphos is acceptable, as illustrated in Figure 5.12. An interesting avenue of future work involves optimizing the data structures used in local databases and their associated query processing algorithms. For example, if each virtual fingerprint is treated as a document and each feature is considered a keyword, then inverted indexes [50] can be used to increase query throughput.

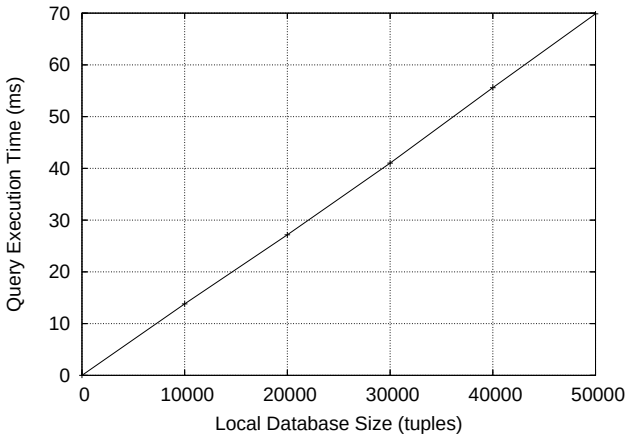

Figure 5.10: Query execution time for databases ranging in size from 0–50,000 tuples.

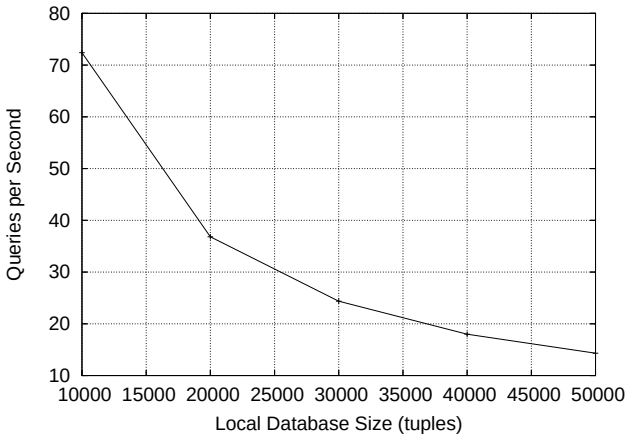

Figure 5.11: Query throughput for reputation databases ranging in size from 10,000–50,000 tuples.

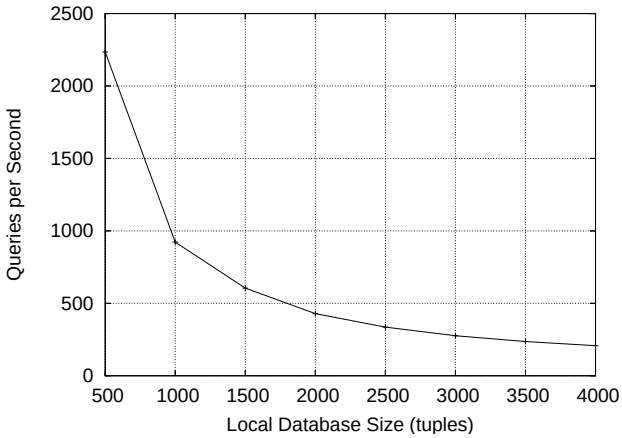

Figure 5.12: Query throughput for reputation databases ranging in size from 500–4,000 tuples.

5.5.5 Concluding Remarks

In this section, we analyzed the performance and utility of Xiphos by simulating a number of grid-computing systems of various sizes. We found that when using an extremely conservative tuple eviction policy, the average size of a local reputation database in a network with 10,000 users was approximately 1,500 tuples after a simulated 5,000 days. In a network of 70,000 users, the average local database contained 4,000 tuples after 5,000 simulated days. When executing a prototype Xiphos implementation on a 1.6GHz laptop, Xiphos could process queries on databases of these sizes at throughputs of 600 and 200 queries per second, respectively, without indexing. The use of a more aggressive, though still reasonable, tuple eviction policy resulted in query throughputs of over 2,200 queries per second on both simulated networks, again without indexing; it is unlikely that the network characteristics of actual grid computing systems would even allow queries to arrive at such a high rate. We also verified that a suitable $\gamma(\cdot)$ function can limit the damage caused by attackers in the system. These observations indicate that Xiphos can be used as a reasonable means of reputation establishment in the open systems of the future, despite the complications arising from the fact that users can legitimately have multiple virtual fingerprints.

In this chapter, we described the use of virtual fingerprinting as the basis for one *particular* reputation system. However, the reputation scores bound to virtual fingerprints can be aggregated according to *any* reputation calculation method, provided that the complications arising from the legitimate assumption of multiple identities (in the form of disjoint virtual fingerprints) are addressed. In particular, systems need to mitigate the effects of malicious users assuming multiple identities to over-influence the system. Additionally, the fact that queries may overlap multiple tuples could lead to problems at naive centralized servers wishing to maintain precomputed reputation scores. The ontology-based definition of the $\gamma(\cdot)$ function discussed in Section 5.5.2 prevents malicious entities from over-influencing our simulated grid computing system; similar definitions are likely to be possible in other domains as well. We also presented a method for maintaining precomputed reputation estimates which could be used to enhance the performance of a centralized deployment of Xiphos. Similar modifications could be made to current and future reputation systems, thereby enabling them to use virtual fingerprints as a means of identity and extending their applicability to attribute-based trust management systems.

5.6 Summary

In this chapter, we examined one of the major systems challenges associated with adopting decentralized ABAC approaches to authorization. In particular, we presented a method for the linking and correlation of multiple identities in attribute-based trust management systems. We discussed how the descriptions that one entity learns about another can be transformed into opaque *virtual fingerprints*, which can be used in place of more traditional user identities as a privacy-preserving basis for audit and incident response systems, anomaly detection algorithms, collusion detection/prevention mechanisms, and reputation systems. To evaluate the utility of virtual fingerprints, we developed the Xiphos reputation system, which we then studied in detail. We presented several deployment models of the Xiphos system, discussed the privacy versus utility trade-off for each of these deployments, and examined the impacts of several attacks against the Xiphos system.

The performance of this system and its costs of deployment were then analyzed in the context of a grid computing scenario. Our evaluation of the Xiphos system indicates that it is an acceptable means of reputation establishment for open systems. In addition, we argued that the more general notion of virtual fingerprints can be used in conjunction with *any* reputation calculation mechanism, thereby allowing reputation systems which rely on more traditional notions of identity to be used in attribute-based trust management systems. Virtual fingerprinting can also be used as the foundation for other useful security services, such as secure audit and incident response systems. These types of systems could be used to ensure that users are held accountable for their actions and to aid in discovering certain types of collusion occurring at points distributed across an open system.

6 TrustBuilder2: An Architectural Framework for Trust Negotiation

Rome ne fut pas faite toute en un jour.

—French proverb

To date, research in trust negotiation has focused mainly on the theoretical issues involved in the trust negotiation process. Most implementations have been designed largely as proofs of concept and, therefore, were never intended to be used heavily in practice. These theoretical works and proofs of concept have been quite successful, and thus researchers must now begin to address the systems constraints that act as barriers to the deployment of these systems. The effort required to develop and optimize the myriad trust negotiation formulations present in the literature can be greatly reduced by first creating a suitable architectural framework that can be parameterized to support various types of trust negotiation systems and components. In this chapter, we present TrustBuilder2, a fully-configurable framework for prototyping and evaluating trust negotiation systems. We then use this framework to examine several interesting systems properties of the trust negotiation process.

TrustBuilder2 leverages a plug-in based architecture, extensible data type hierarchy, and flexible communication protocol to provide a framework within which numerous trust negotiation protocols and system configurations can be quantitatively analyzed. We discuss the design and implementation of TrustBuilder2, present several case studies highlighting the framework's extensibility, and examine the costs associated with designing flexible authorization systems. We also study the performance of TrustBuilder2, explore the bottlenecks of the trust negotiation process, and leverage this knowledge to identify important topics for future research as well as to identify a novel method for attacking trust negotiation systems.

6.1 Introduction

Previous research in trust negotiation has been primarily of a theoretical nature, focusing on a number of important issues including languages for expressing re-

source access policies (e.g., [17; 19; 57; 83]), protocols and strategies for conducting trust negotiations (e.g., [20; 68; 69; 122]), and logics for reasoning about the outcomes of these negotiations (e.g., [24; 119]). These results provide a strong theoretical foundation upon which provably-secure authorization systems can be designed, built, and verified. Some of the techniques discussed in the trust negotiation literature have also been shown to be viable solutions for real-world systems through a series of implementations (such as those presented in [20; 58; 67; 118]) that demonstrate the feasibility of using these theoretical advances. However, after several years of research, trust negotiation protocols have yet to make their way into the mainstream.

Prior to deploying access control systems based on trust negotiation, the systems and architectural properties of trust negotiation must be more fully understood. Existing trust negotiation implementations have been developed largely as proofs of concept designed to illustrate the feasibility of the underlying theory and have performed admirably in this capacity. Unfortunately, these proof-of-concept implementations can be difficult to configure and use and are generally not easily extended or modified. As a result, exploring certain types of systems research problems surrounding trust negotiation becomes difficult. For example:

- Is it possible to unify the myriad formulations of trust negotiation described in the research literature under a common *framework*? Adopting such a framework would make it possible to further deploy and experiment with novel trust negotiation systems and components in a grassroots fashion.
- What are the performance bottlenecks of the trust negotiation *process*, as opposed to those of a specific *implementation*? How can we quantify these costs?
- How can we identify and measure the severity of attacks that are made possible by various approaches to trust negotiation?
- When all other factors are held constant, what are the costs and benefits of using one trust negotiation system component (e.g., negotiation strategy, policy compliance checker, etc.) over another? To what extent do various approaches limit or mitigate attacks?

In an effort to address these types of systems research challenges, we have developed TrustBuilder2, a flexible and reconfigurable Java-based framework for supporting trust negotiation research. TrustBuilder2 supports a plug-in based architecture to allow *any* system component to be modified or replaced by users of the system without requiring modification or recompilation of the underlying framework. TrustBuilder2 is also agnostic with respect to the formats of credentials and

policies used during the negotiation. Support for new policy languages, credential formats, or trust negotiation evidence types (e.g., trust tickets [20], uncertified claims [20; 24], or proof fragments [119]) can be incorporated by implementing extensions to the TrustBuilder2 data type hierarchy. In this chapter, we discuss the design and implementation of TrustBuilder2, as well as the results of research carried out using this framework. Specifically, we make the following contributions:

- TrustBuilder2 represents the first fully-configurable framework for trust negotiation. TrustBuilder2 leverages a plug-in based architecture, extensible data type hierarchy, and flexible communication protocol to provide a framework within which numerous trust negotiation protocols and system configurations can be quantitatively analyzed.
- We carry out performance studies using TrustBuilder2 to help identify the primary performance bottlenecks of the trust negotiation process. This leads to the identification of a novel class of denial of service attacks against trust negotiation systems that exploits the time required to check compliance with trust negotiation policies. This differs significantly from the attacks previously discussed in the research literature, which focus predominantly on the cryptographic aspects of the trust negotiation process.
- TrustBuilder2 demonstrates that adding a high degree of flexibility to advanced authorization frameworks does not necessarily lead to high runtime overheads. In Section 6.7, we show that the time spent handling the indirection needed to support user plug-ins and other extensions amounts to less than 0.2% of the total execution time.
- The insights gained while studying the performance of TrustBuilder2 also inspired novel research leading to the development of CLOUSEAU, a highly-optimized trust negotiation policy compliance checker, which will be discussed in greater detail in Chapter 7.

The remainder of this chapter is organized as follows. In Section 6.2, we examine previously-developed trust negotiation implementations and discuss the features provided by these systems. Section 6.3 presents a use case analysis examining the characteristics of several target deployment domains for trust negotiation systems. In Section 6.4, we use the results of our use case analysis to identify a number of useful features that should be provided by frameworks designed to facilitate research on the systems aspects of trust negotiation and the eventual deployment of authorization systems based on trust negotiation; we then identify the subsets of these desiderata that are addressed by existing trust negotiation implementations. Section 6.5 presents the architecture of the TrustBuilder2 framework for trust nego-

tiation. In Section 6.6, we explore the ways in which TrustBuilder2 can be extended. Section 6.7 discusses a performance evaluation of the TrustBuilder2 framework and lessons learned through this process. In Section 6.8, we examine how TrustBuilder2 addresses the desiderata presented in Section 6.4. We also discuss attacks on trust negotiation systems, potential research topics uncovered by our performance evaluation, and describe how to obtain the TrustBuilder2 framework. We then present our conclusions in Section 6.9.

6.2 Existing Implementations

Over the last several years, several implementations of trust negotiation systems have been described in the literature. The earliest such implementation was the TrustBuilder architecture for trust negotiation [118]. TrustBuilder is a Java implementation that supports the use of X.509 certificates to encode attributes and XML to represent policies written using the IBM Trust Policy Language (TPL) [57]. The IBM Trust Establishment (TE) compliance checker is used to determine whether a certain set of credentials satisfies a given policy. TrustBuilder has been embedded into an implementation of TLS [58] and several other protocols to demonstrate the applicability of trust negotiation in existing systems. TrustBuilder supports the use of only one credential format, one policy language, and one trust negotiation strategy.

Trust-$\mathcal{X}$ [20] is an XML-based framework for supporting trust negotiations in peer-to-peer systems. In Trust-$\mathcal{X}$, each user creates an $\mathcal{X}$-profile that stores $\mathcal{X}$-TNL certificates describing their attributes along with uncertified declarations containing other information about the user (e.g., preferences, phone numbers, or other such information). To the best of our knowledge, Trust-$\mathcal{X}$ does not support credential formats other than $\mathcal{X}$-TNL certificates nor policies specified in any language other than $\mathcal{X}$-TNL. To allow users to optimize various aspects of the trust negotiation process, Trust-$\mathcal{X}$ supports a variety of interchangeable trust negotiation strategies. Another particularly innovative feature of the Trust-$\mathcal{X}$ framework is its support for *trust tickets*. Trust tickets are receipts that attest to the fact that a user recently completed a negotiation with another party. These trust tickets can then be presented within some limited lifetime (typically 24-48 hours) to bypass redundant portions of future negotiations with the same party.

In [67], Koshutanski and Massacci describe a trust negotiation framework designed for web services. This framework facilitates the composition of access policies across the constituent pieces of a workflow, the discovery of credentials needed

to satisfy these policies, the management of the distributed access control process, and the logic to determine what missing credentials must be located and provided to satisfy a given policy. The use of X.509 and SAML credentials is supported by the framework, as is the use of the negotiation strategies described in [67] and [69]. Policies are represented using a Datalog-based language. To the best of our knowledge, the use of other credential formats, negotiation strategies, or policy languages is not supported.

In [38], De Coi and Olmedilla describe a flexible and expressive trust negotiation implementation. The authors examined the PEERTRUST [98] and PROTUNE [25] systems in an effort to derive a set of common requirements that should be supported by any trust negotiation implementation, and then implemented a framework embodying these requirements. Their system supports PEERTRUST and PROTUNE inference engines, and allows users to add support for other inference engines. Furthermore, users can specify trust negotiation strategies as *action selection algorithms* within their framework. Credentials are expressed as signed logical statements and are loaded from a credential repository that is accessed by their implementation. To the best of our knowledge, the use of other credential formats is not supported.

While not specifically an implementation of a trust negotiation framework, Cassandra [17] is a policy language for distributed access control that supports the specification of policies with a tunable level of expressiveness. The features of Cassandra are such that it can encode a certain, fixed, trust negotiation strategy. A prototype system that uses the Cassandra language has been implemented in OCaml to facilitate research on the features of this policy language. This implementation uses a single computer to simulate the interactions of up to thousands of users in a distributed system and has been used to explore the use of Cassandra within the medical records domain [18]. Support for new constraint domains for the Cassandra language or improved policy evaluators can be added by implementing plugins for the system. Since this implementation is actually a simulator, credentials are represented as text strings, rather than concrete certificates protected by digital signatures.

6.3 Use Case Analysis

To derive the functional requirements for the TrustBuilder2 system, we examined three interesting potential usage scenarios for decentralized ABAC systems. In particular, we explored scenarios involving client-server information sharing on

the World Wide Web, scientific grid computing, and information sharing in high-assurance environments. It should be noted that most research to date has focused on two-party trust negotiations. In this section, however, we relax this assumption and consider cases in which multi-party interactions could be useful, in hopes of deriving requirements that will lead to trust negotiation architectures with increased levels of flexibility.

6.3.1 The World Wide Web

The World Wide Web is an almost endless source of examples of client-server interactions ranging from e-commerce to research applications. Rather than examining a particular client-server interaction in detail, we will discuss this problem more generally. Consider the case in which some service provider wishes to offer their service, S, to any clients who satisfy some access control policy, P, which is specified in any one of a number of trust negotiation languages (e.g., Cassandra [17], $\mathcal{X}$-TNL [19], TPL [57], or RT [83]). Upon requesting access to S, the client and the server begin a trust negotiation to determine whether the client satisfies P and should be granted access to the service.

In such interactions, the client and server are likely to have different requirements governing the execution of the trust negotiation. From the client's perspective, the following assumptions may hold true:

- To prevent identity theft, the client is likely to have few credentials which are freely disclosable. That is, most of the client's credentials will be protected by release policies.
- Clients will not often be concerned with the threat of denial of service (DoS) attacks being launched on their trust negotiation agents by the services that they contact. However, information gathering attacks will be expected.
- To ensure access to S, the client will often be willing to do extra work. For instance, if a client does not possess a credential required by P, they may be willing to try to locate the missing credential (within certain limits).
- Technically savvy clients may wish to be involved directly in the trust negotiation process. This could take the form of a client either writing his or her own credential release policies or making strategic decisions at the time of negotiation. Naive clients may wish to have "reasonable" policies provided to them and allow their user agent to perform the entirety of the negotiation process using these policies.

In contrast to these points, the following assumptions are likely to hold true at the server:

- Many servers are public interfaces to resources and will likely have very few credentials protected by release policies.
- Due to the heavyweight nature of trust negotiation, servers will be concerned with the threat of DoS attacks. However, information gathering attacks will not be a concern, as most credentials held by the server will be publicly available.
- Due to the threat of DoS, servers are unlikely to engage in expensive external efforts to ensure the success of a negotiation (such as searching for requested credentials).
- The negotiation protocol executed by the server must be entirely automated, as there is no human operator present to make decisions at runtime.
- To increase revenue or information dissemination, servers will likely wish to have a robust trust negotiation configuration which will maximize the number of clients who can access the system while minimizing the cost of a negotiation to the server.

In examining these points, we notice two main discrepancies that exist between the client and server. First, servers wish to maximize the number of trust negotiations that can be conducted per unit time, while the client is often willing to allow this process to take longer in hopes of ensuring success. Second, the fact that servers are public entities is in direct contrast with the private nature of individuals. Several interesting features will be derived from these differences later in this chapter.

6.3.2 Scientific Grid Computing

In scientific grid computing, researchers utilize geographically and administratively distributed resources to solve complicated research and simulation problems. For example, a scientist in Utah might wish to user her allocations of processor time on computing clusters in Illinois and California, access a wave tank owned by a university in Florida, and use a data set collected by colleagues in Maine to analyze various characteristics of a tsunami. In order to successfully submit such a job for execution on a grid, the scientist must establish the right to access resources owned by multiple resource providers.

The nature of the interactions that must occur to successfully negotiate for access to a computational grid lead us to make the following assumptions regarding trust

negotiation:

- The integrity of jobs submitted to these systems is of the utmost importance to users, as resource constraints and deadlines often discourage running a job multiple times. Users will likely have stringent requirements that must be satisfied by resource providers and may even require interaction with third-party reputation services to ensure that the results that they obtain will be accurate.
- The time required to run many grid computing jobs is measured in hours or days. This implies that users will be willing to spend extra time and effort to ensure the success of their trust negotiation sessions, as this cost can be amortized over the duration of the job.
- Even though users may be willing to interact with third parties to help ensure the success of their negotiations, this must be done with care, as third parties may give incorrect answers either accidentally or maliciously (particularly if there is a high demand for resources, as there tends to be near deadlines).
- Users will have many sensitive credentials detailing their research and organizational memberships, especially if they are employed in government or industry research labs where need-to-know is used to limit information dissemination.

Though at first glance the resource providers in computational grids seem equivalent to the servers discussed in the World Wide Web use case, they are in fact quite different. Resource providers in computational grids are likely to make the following assumptions regarding trust negotiation:

- Maximizing system utilization is key. Many resources attached to computational grids are expensive to own and operate, thus it is in the best interest of resource providers to ensure that the maximum number of authorized users gain access, to help pay the cost of operating the resources. This implies that resource providers will be configured to maximize the number of clients able to access the system successfully and will be concerned less with the speed with which trust negotiation sessions occur.
- The length of time required to run a grid job implies that resource providers will be carrying out fewer trust negotiation sessions than, for example, e-commerce web sites. As a result, resource providers will be more open to the idea of doing extra work to ensure the success of a negotiation. For instance, resource providers may attempt to locate credentials requested by a particular client.

- Resource providers will likely have some number of public credentials, as was the case with servers on the World Wide Web, though they will likely have other more sensitive credentials that will be protected by release policies (particularly in government computing environments).

While the grid computing application domain has some similarities with the Web environment, the inherently collaborative nature of this environment gives rise to several unique assumptions. These contrasting assumptions will elicit requirements for features not needed in the client-server domain.

6.3.3 High Assurance Environments

High-assurance environments such as disaster management networks or critical infrastructures, such as the electric power grid, tend to have stringent requirements on information sharing. In particular, it is of the utmost importance that *qualified* individuals gain access to all necessary information in a timely manner, even if those qualified individuals happen to be *outsiders*. For instance, during a time of crisis, it is important that police, fire, and rescue workers can access status information about the incident to which they are responding, even if they happen to be volunteers from another district. In the case of the electric power grid, there are times when market-sensitive information should be released to competing entities to enable proper response to changing conditions in the grid and avoid cascading blackouts or other failure situations.

As with the previous two use cases, we examine characteristics of this environment from the perspective of both clients (information consumers) and servers (information producers). From a client viewpoint, the following assumptions are likely to hold true:

- Emergencies and other situations in which information sharing is likely to occur in high-assurance environments are usually highly context-dependent. The variability that this introduces can make it difficult for users to understand why their access requests are permitted or denied. The reasons for the success and failure of any access control decisions should be explained carefully by the underlying trust negotiation architecture.
- Often critical decisions must be made quickly. Users will not tolerate long delays when requesting information in this environment.
- The criticality of these systems implies that users want their interactions to succeed if at all possible. Interactions with external parties that would increase the likelihood of a successful negotiation will be tolerated, though they

will be subject to stringent timing constraints.

- Users will likely act in uncharacteristic ways during times of crisis or when adverse conditions arise in the system. Since these environments have stringent security requirements, atypical user activities carried out during a crisis may be investigated after the crisis is past. To protect themselves, users will require that verifiable audit trials be stored. These records should include information about both the external environment (i.e., system context) and specific records detailing the negotiations carried out by the user.

With respect to information producers in high-assurance environments, the following assumptions regarding information sharing are likely to hold true:

- The high-security nature of these environments implies that in many ways they are more constrained than the World Wide Web or grid computing environments discussed previously. Support for a wide array of configuration options regarding supported credential types and strategies is likely to be unnecessary; a preordained set of these options should suffice.
- Since flow of information is key to the operation of these environments, information providers will want their trust negotiation systems be fast and resistant to denial of service attacks.
- In general, high-assurance environments have very stringent requirements on the flow of information. However, under certain circumstances (e.g., emergency conditions) information providers may take a more permissive approach to information dissemination in hopes of solving problems of critical importance. To help detect patterns of abuse, information providers will need strong audit trails detailing exactly what information was released and to whom it was released.

6.4 System Requirements

We now examine the assumptions made in the previously presented use cases and derive functional requirements for a general-purpose trust negotiation framework. These requirements fall naturally into three categories: requirements placed on core trust negotiation components, requirements relating to external interactions, and performance and extensibility requirements. In all cases, we examine both the derivation and implications of each requirement. We first discuss the requirements relating to the general functionality afforded by the core components of the trust negotiation system.

Arbitrary policy languages In many cases, resource providers will wish to be accessible to as many potential clients as possible. To facilitate this, these entities should be able to parse access policies written in a variety of formats (e.g., Cassandra [17], $\mathcal{X}$-TNL [19], TPL [57], RT [83], and XACML [93]). It should be easy to add support for new policy languages to deployed systems.

Arbitrary credential formats To further enable interactions with a maximal set of users, trust negotiation systems should support the use of multiple credential formats such as X.509 certificates [61] and SAML assertions [30]. It should also be easy to add support for new credential formats to deployed systems.

Interchangeable negotiation strategies Trust negotiation is by nature a strategy-driven process. Entities should be able to choose negotiation strategies that direct the execution of a trust negotiation session to meet their particular goals (e.g., maximizing privacy or minimizing latency). One can imagine many situations in which the goals of the participants in a negotiation might be conflicting. The use of families of interoperable strategies that allow negotiation participants to choose different, yet compatible, strategies (e.g., as in [122]) should be supported. It should be possible to add support for new negotiation strategies to deployed systems.

Flexible policy and credential stores Clients are likely to utilize several computing devices—such as desktop computers, laptops, PDAs, and smart phones—during the course of their daily activities. It is therefore important that a trust negotiation architecture support interactions with a variety of flexible policy and credential stores (e.g., [7; 111]) that will enable users to effectively manage their digital identities across multiple devices.

While these basic flexibility requirements are important, they do not address all aspects of the negotiation process. In particular, we must also consider the ability to add more advanced features that might increase the efficiency, understandability, or functionality of the trust negotiation process. For instance, the World Wide Web use case highlighted the need for users to be able to request help in locating missing credentials from external entities (e.g., as in [119]). Similarly, the grid computing use case introduced the need to support a wider array of helpful third parties, such as reputation systems. In yet other cases, the active participation of the human on whose behalf a negotiation is initiated may be desired. Unfortunately, the naive inclusion of these features can lead to a variety of problems. The following requirements help to safely enable beneficial external interactions:

Strategy-driven external interactions Negotiation participants should have the ability to interact with a wide range of external entities that can help solve

difficult problems which may arise during the negotiation. Examples of such interactions include the calculation of reputations or credential chain discovery.

Advanced logging capabilities The architecture should include a logging service that can record information regarding any aspect of the negotiation process. Since a high degree of logging is not always needed, the logging subsystem should support the recording of logs at various granularities.

Tunable human involvement In some instances, humans may wish to be involved directly in the negotiation process. For example, users may want to specify an "ask me" release policy for a sensitive credential, see a visual representation of the negotiation process for policy evaluation purposes, or be involved in the decision-making process when the negotiation comes to a point where there are multiple execution paths that could be followed rather than relying on a predefined strategy. The framework should support extensions that can add a human "in the loop" if such features are requested.

Selective feature activation To enable more efficient or more secure trust negotiation sessions, the features enabled by the framework should be fully configurable. For instance, disabling support for visualization features and external interactions might increase the performance of the system, while disabling third-party plug-ins might increase overall system security and trustworthiness.

Feature ordering To enhance the performance of the system and its robustness against attack, entities should have the ability to choose the order in which certain functionalities are invoked. For instance, it should be possible for a negotiation strategy to choose the time at which credentials are validated. That is, there may be benefits to validating credentials as they become relevant, rather than validating them as they are received.

The diversity of use cases that we considered leads us to believe that it represents a useful set of features to support when designing a general-purpose trust negotiation framework. However, the requirements presented above cannot be considered complete, as it is impossible to consider every possible trust negotiation use case. To acknowledge and partially address this gap, we introduce one further requirement that helps ensure additional features can be easily added to the framework.

Extensibility The framework must support the addition of new functionality after deployment without requiring modifications to the existing code base. Example features may include (but are not limited to) the inclusion of new

	TrustBuilder	Trust-$\mathcal{X}$	Koshutanski	De Coi	Cassandra
Arbitrary policy languages	N	N	N	P	P
Arbitrary credential formats	N	N	P	N	N
Interchangeable negotiation strategies	N	P	P	Y	N
Flexible policy and credential stores	N	N	N	N	N
External interactions	N	N	N	Y	Y
Tunable human involvement	N	N	N	N	N
Advanced logging	N	N	N	P	N
Selective feature activation	N	N	N	N	N
Feature ordering	N	N	N	N	N
Extensibility	N	N	N	P	P

Table 6.1: Features supported by existing trust negotiation implementations (Y = yes, N = no, P = partially supported).

local data processing rules, the enforcement of obligations, and the incorporation of new data types (e.g., trust tickets, partial proofs, or other forms of evidence) into the negotiation process.

Table 6.1 identifies the subsets of these requirements addressed by each of the trust negotiation frameworks discussed in Section 6.2. As shown, no existing trust negotiation framework provides even partial support for more than half of the identified features; this is not surprising, given that these implementations were not meant to be general-purpose frameworks. We will revisit these requirements in Section 6.8 to discuss the ways in which TrustBuilder2 addresses these desiderata.

6.5 The TrustBuilder2 Framework

In this section, we describe the design of TrustBuilder2, a Java-based framework for trust negotiation. The primary goal in designing TrustBuilder2 was not to implement one particular trust negotiation protocol, but rather to provide a framework within which any number of trust negotiation techniques can be implemented and evaluated. This led to unique challenges in designing the communication protocol used by negotiation participants, the data type hierarchy used by TrustBuilder2, and the software architecture of the system. In this section, we describe the above facets of the TrustBuilder2 framework.

6.5.1 Communication Protocol and Data Types

One of the first challenges we faced when designing TrustBuilder2 was defining a communication protocol that could be interpreted by the framework, without constraining the trust negotiation protocols that could be supported. For example, we did not want to mandate that *only* credentials and policies are exchanged during a trust negotiation session, as that would prevent the implementation of pro-

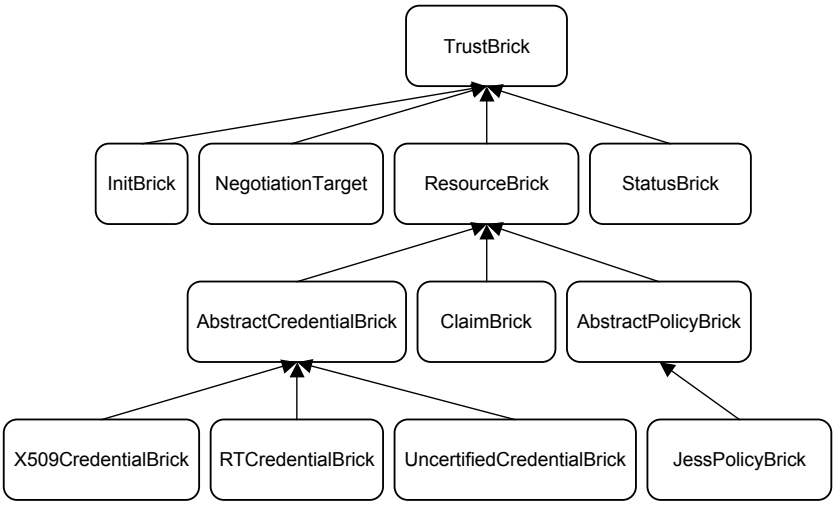

Figure 6.1: Class hierarchy for several important TrustBrick subclasses.

tocols such as Trust-$\mathcal{X}$ [20] and PeerAccess [119] within the TrustBuilder2 framework, since these protocols also exchange digitally-signed trust tickets and proof-fragments, respectively. To this end, TrustBuilder2 uses a very simple communication protocol combined with an extensible data type hierarchy to enable the implementation of a wide range of trust negotiation protocols.

Data type hierarchy

At a high level, a trust negotiation session is an exchange of messages containing credentials, policies, uncertified claims, and other information between two parties. In order to support the widest possible range of trust negotiation protocols, the core components of the TrustBuilder2 framework (which will be described in Section 6.5.2) rely heavily on the use of an extensible data type hierarchy. All types of information that might be exchanged between negotiating parties are represented as subclasses of the TrustBrick class, which forms the basic building block of the trust negotiation process. In this way, users can extend the data types supported by TrustBuilder2 without modifying each component in the system; components can simply ignore TrustBricks that they do not know how to process, leaving them for other system components to handle. Entities then exchange TrustMessage objects containing one or more of these TrustBricks.

Figure 6.1 shows the relationships between several important subclasses of TrustBrick. The InitBrick, NegotiationTarget, and StatusBrick classes are used to provide high-level information regarding a negotiation: InitBricks are used to establish the parameters of a trust negotiation, a NegotiationTarget is used to indicate the par-

ticular resource that the initiator of a trust negotiation wishes to access, and a StatusBrick is included in the last message of the negotiation to indicate whether the negotiation succeeded in establishing trust between the participants. Any item exchanged during a trust negotiation that could possibly be protected by a release policy is a subclass of ResourceBrick. This ensures that TrustBuilder2 can properly enforce disclosure requirements on data items without necessarily understanding the data item itself.

The AbstractCredentialBrick and AbstractPolicyBrick classes are used to represent attribute certificates and policies at an abstract level, which enables components of TrustBuilder2 to handle credentials and policies of various formats without needing to understand the intricacies of each format explicitly. The X509CredentialBrick class is used to hold information about X.509 certificates, while the RTCredentialBrick class holds information about RT credentials [83]. The UncertifiedCredentialBrick class provides TrustBuilder2 with the ability to create "fake" credentials on-the-fly to facilitate the rigorous testing of system components as they are developed. Lastly, the JessPolicyBrick class is used to hold policies that can be interpreted by the CLOUSEAU compliance checker, which we have developed to explore efficiency issues in the design of trust negotiation compliance checkers. CLOUSEAU will be discussed in greater detail in Chapter 7.

In Section 6.6, we illustrate the ways in which this extensible type hierarchy facilitates the extension of the TrustBuilder2 framework to incorporate new features, such as support for new policy languages or credential types. Readers interested in more detail regarding TrustBrick or its subclasses should consult the TrustBuilder2 programmer documentation included with the TrustBuilder2 distribution.

The communication protocol

As previously mentioned, the TrustBuilder2 communication protocol is nothing more than an exchange of TrustMessage objects containing one or more TrustBricks between the participants of the negotiation. The first message sent by the initiator of the trust negotiation session contains a single InitBrick object describing the TrustBuilder2 system configurations (i.e., strategy families, credential formats, and policy languages) that she supports, along with other system parameters. If the responder supports a system configuration that is compatible with one of the system configurations proposed by the initiator, he returns a TrustMessage containing another InitBrick describing this system configuration. At this point, both parties can configure their TrustBuilder2 framework to use this mutually-acceptable system configuration during their negotiation session. The initiator then responds with a

TrustMessage containing a NegotiationTarget that indicates the resource that she wishes to access. Beyond this, no constraints are imposed on the contents of these messages; future TrustMessage objects exchanged by the participants are handled by the strategy modules (described in the next section) supported by each of the participants, rather than the core TrustBuilder2 framework. This allows TrustBuilder2 to support a wide range of trust negotiation protocols without requiring protocol-specific modifications be made to the framework itself.

6.5.2 Software Architecture

Figure 6.2 presents a high-level architecture diagram of the TrustBuilder2 runtime system. The design choices made when implementing this system are a result of the requirements discussed in Section 6.4 and the need to support the flexible communication protocol discussed in Section 6.5.1. Components enclosed in dashed boxes are not included in the current version of TrustBuilder2; they are only meant to serve as examples of components that could be developed by users as plug-ins and added to the TrustBuilder2 data path. We now describe each of the major components identified in this diagram and comment on the flow of data between components.

The external interface to the TrustBuilder2 runtime system is provided by the TrustBuilder2 class. Trust negotiation sessions are conducted by making a series of calls to methods exposed by this class. When a new trust negotiation session is started, the TrustBuilder2 class creates and manages a Session object that keeps track of all necessary state between rounds of the negotiation. For example, after the exchange of InitBricks described in Section 6.5.1, the Session object will contain a description of the TrustBuilder2 configuration to be used for this session, including the strategy module to use, a list of supported policy languages, and a list of supported credential formats. Any component in the system can add its own internal state to a given Session object. This allows components to avoid maintaining this state locally and eases the development of reentrant system components.

During the trust negotiation process, all incoming TrustMessages are processed by the TrustBuilder2 object, which generates a response TrustMessage to return to the remote participant. Prior to processing a remote TrustMessage itself or dispatching it to the StrategyModuleMediator, the TrustBuilder2 object first passes all incoming messages to the IOManipulationModule. The IOManipulationModule is the first class to process each incoming TrustMessage and the last class to process each outgoing TrustMessage. This component is capable of loading user-defined plug-ins that can examine and modify all TrustMessage objects entering and leaving the Trust-

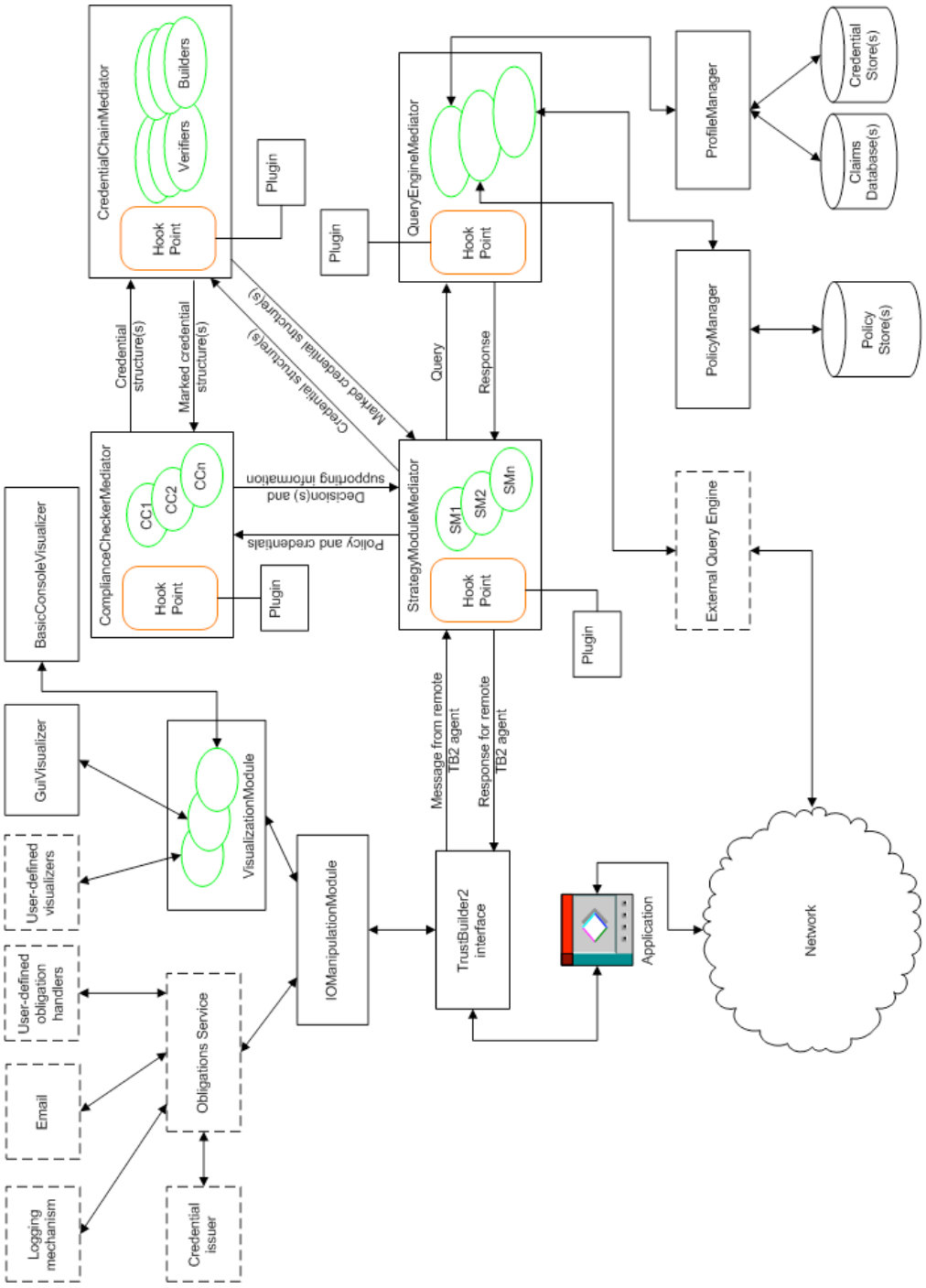

Figure 6.2: TrustBuilder2 architecture overview diagram.

Builder2 runtime system. The VisualizationModule is an example plug-in to the IOManipulationModule that provides an interface for writing and using custom logging and visualization components. The TrustBuilder2 distribution includes two such components: the GuiVisualizer class is a plug-in that uses the Swing API to graphically visualize every TrustMessage processed by TrustBuilder2, while the BasicConsoleVisualizer is a plug-in that provides a console logging facility.

The core of the TrustBuilder2 runtime system—that is, the interfaces to the strategy modules, compliance checkers, credential and policy stores, and credential manipulation routines—is provided by a set of four *mediator* classes. Each mediator class acts as a dispatcher providing access to any number of user-specified trust negotiation system core components as described in more detail below.

StrategyModuleMediator This class acts as the coordinator for a given trust negotiation session, as it dispatches incoming TrustMessages to the appropriate strategy module according to the strategy that was chosen for use during that session; any number of strategy modules can be loaded into the StrategyModuleMediator at a given time. Given an incoming TrustMessage and any state that it had previously stored, a strategy module decides what should be disclosed in a response to the remote party. Strategy modules can make any number of calls to the ComplianceCheckerMediator, CredentialChainMediator, or QueryEngineMediator components during their execution.

ComplianceCheckerMediator The ComplianceCheckerMediator acts as a dispatch routine that enables access to the various compliance checkers loaded into the TrustBuilder2 runtime system. A compliance checker takes as input a policy, a set of credential chains, and a set of claims, and then determines whether the supplied credentials and claims satisfy the given policy. The ComplianceCheckerMediator chooses the compliance checker to use for a given set of inputs based on the language of the policy that its caller is trying to satisfy. TrustBuilder2 supports the use of compliance checkers that find at most one satisfying set of credentials for a given policy and also compliance checkers that find all satisfying sets of credentials for a given policy. More details regarding compliance checker modes of operation will be discussed in Chapter 7.

CredentialChainMediator This component is responsible for creating and verifying chains of credentials used during a trust negotiation. The CredentialChainMediator supports the use of any number of credential chain construction and verification algorithms that can be implemented as user plugins. TrustBuilder2 includes support for one credential chain construction al-

gorithm and two credential chain verification algorithms by default.

QueryEngineMediator This component accepts queries from other system components, processes these queries, and returns a result. All queries are subclasses of the AbstractQuery data type and are dispatched to an appropriate query engine based upon their query type. For example, queries to load a user's credentials from disk are dispatched to the ProfileManager. Any number of query engines can be loaded by the QueryEngineMediator and support for new query engines can be added by developing plug-ins for the QueryEngineMediator. Currently, TrustBuilder2 includes support for two query engines: the PolicyManager and the ProfileManager, which are themselves extensible interfaces to a user's policy stores, and credential and claim repositories, respectively.

For the sake of brevity, not every component of TrustBuilder2 was discussed in this section. Readers desiring a more complete treatment of the components of the TrustBuilder2 system should consult the programmer documentation available in the TrustBuilder2 distribution.

6.5.3 Default Configuration

By default, TrustBuilder2 includes support for a version of the TrustBuilder1-Relevant strategy for trust negotiation described in [122] that has been modified to further minimize information disclosure in the event that multiple satisfying sets of credentials are found for a given policy during a negotiation. As described above, TrustBuilder2 supports the use of X.509 v3 credentials during interactions with remote parties but can also use uncertified "test" credentials to exercise the functionality of new plug-ins or components as they are being designed and developed. Plug-ins are provided for the CredentialChainMediator that allow TrustBuilder2 to form a set of credential chains from a collection of credentials of *any* format and to verify the authenticity of the credential chains that were formed. TrustBuilder2 currently supports the CLOUSEAU compliance checker and can load a user's policies, credentials, and uncertified claims from repositories on the local file system.

6.6 Case Studies in Extensibility

In this section we discuss the extensibility of TrustBuilder2 at a high level, as well as provide a more detailed treatment of two significant extensions added to the

framework after its initial development.

6.6.1 General Extensibility

As was our goal from the outset, almost every component of the TrustBuilder2 framework can either be extended or replaced by a user-defined plug-in. Because the TrustBuilder2 framework was developed using Java, dynamic class loading can be used to incorporate these user plug-ins at runtime without requiring any modification to the TrustBuilder2 framework itself. Extensions to the primary components of TrustBuilder2—that is, the IOManipulationModule, StrategyModuleMediator, ComplianceCheckerMediator, CredentialChainMediator, and the QueryEngineMediator—as well as plug-ins that interpose between these components, can be added to the system quite easily. Users simply write and compile plug-ins conforming to the appropriate interfaces and instruct the TrustBuilder2 runtime system to incorporate these modules the next time that a TrustBuilder2 object is created. For instance, adding a new strategy to TrustBuilder2 involves writing a class implementing StrategyModuleInterface and adding this class to the list of strategy modules to be loaded by the StrategyModuleMediator.

We now discuss how the abstract type hierarchy used by TrustBuilder2 allows support for new credential and policy formats—as well as new forms of negotiation evidence—to be added to the the system without requiring modifications to the underlying framework. As will be shown, this process is very straightforward and allows support for novel trust negotiation features to be easily incorporated into the TrustBuilder2 framework.

6.6.2 X.509 Credentials and Uncertified Claims

Our initial version of the TrustBuilder2 framework only included support for uncertified "test" credentials, as encoded by the UncertifiedCredentialBrick class. These credentials can be easily created and modified and thus allow for rapid and efficient testing of system components. However, to better study the properties of trust negotiation systems that might be deployed in practice, support for more realistic credential types was required. As a result, we added support for uncertified claims encoding user data such as phone numbers or preferences (as in [20; 24]), as well as X.509 v3 certificates to the TrustBuilder2 framework.

Support for uncertified claims required two extensions to TrustBuilder2. First, the ClaimBrick data type was added as a subtype of the ResourceBrick data type in the TrustBuilder2 type hierarchy (see Figure 6.1). Subtyping ResourceBrick in this way

ensures that uncertified claims can be treated as sensitive and optionally protected by release policies. Second, a loader plug-in was written for the ProfileManager so that uncertified claims could be loaded from the file system. In total, less than 300 lines of commented code had to be written to support the addition of uncertified claims to TrustBuilder2.

Adding support for X.509 v3 certificates to TrustBuilder2 was accomplished in a similar manner. Specifically, the X509CredentialBrick data type was added as a subtype of the AbstractCredentialBrick type and another loader plug-in was written for the ProfileManager so that X.509 certificates could be loaded from the file system. The X509CredentialBrick data type wraps the functionality of the X.509 data type supported by Java natively and provides additional methods that extract attribute information from a credential's extension OID fields, populate the data structures used by AbstractCredentialBrick objects, create and verify proof of ownership challenges, and verify issuer signatures. Since TrustBuilder2's default policy compliance checker, credential chain construction algorithms, and credential chain verification algorithms operate on AbstractCredentialBrick objects, no further modifications were needed for TrustBuilder2 to support X.509 v3 certificates. Fewer than 1000 lines of commented code were needed to implement the plug-ins for this support.

6.6.3 RT Credentials and Policies

To further extend the functionality of TrustBuilder2, we have also implemented support for RT_0 and RT_1 credentials and policies [83]. Adding support for the necessary credential types involved a process similar to that followed for supporting X.509 credentials. That is, TrustBuilder2's type hierarchy was extended to include RT credentials, and a loader plug-in was written to read these credentials from disk. However, since Java does not support RT credentials natively, considerably more code had to be written than was the case for adding support for X.509. In total, approximately 3500 lines of commented code were required to add support for loading, parsing, and using these types of credentials within the TrustBuilder2 framework.

In general, adding support for a new policy language to TrustBuilder2 would require developing a new policy compliance checker that is capable of analyzing the satisfaction of this new type of policy. Such a compliance checker would take the form of a plug-in to the ComplianceCheckerMediator. However, this was not necessary for RT_0 and RT_1 policies. As we will see in Chapter 7, these types of policies can be compiled into a format that can be efficiently analyzed by the CLOUSEAU

compliance checker, which is already supported by TrustBuilder2. Currently, policies must be compiled in an offline manner prior to being used by TrustBuilder2, which limits the credential chain discovery functionality supported by RT. To overcome this barrier, we plan to implement a plug-in that interposes between the StrategyModuleMediator and the ComplianceCheckerMediator and compiles RT policies at runtime.

6.7 Performance Evaluation and System Profiling

We now discuss the results of a preliminary performance evaluation of the TrustBuilder2 framework. Our primary goals in this investigation were to evaluate the overheads associated with the flexible nature of TrustBuilder2 and to better understand the bottlenecks involved in the trust negotiation process. As TrustBuilder2 relies heavily on dynamic class loading and the use of indirection through mediator classes to trigger the execution of various plug-ins installed in the system, it seemed likely that the flexibility afforded by TrustBuilder2 might come at the cost of high overheads. Also, by understanding where time is spent during the trust negotiation process, we can better allocate our limited development time to focus on problem areas in the implementation.

6.7.1 The Scenario

Our scenario was designed to be a realistic trust negotiation scenario that might take place in one branch (Acme Springfield) of a national-scale corporation (Acme Fabrication). In this scenario, an employee wants to access a file server containing sensitive files related to "Project X." The policy protecting the Project X file repository states that an authorized entity must be either (i) a full-time employee of Acme Springfield who has an Acme Fabrication issued sensitive document training certification and works in a department whose number is in the range 2460–2469, or (ii) a full-time employee of Acme Springfield who has an Acme Fabrication issued sensitive document training certification, works in a department whose number is in the range 2400–2499 and was granted an "access exception" for Project X by either Alice or Bob. This policy was thought to be a reasonable example of a negotiation that one might see in a large corporation, as it is much simpler than managing a long access control list, but also includes provisions for the explicit white-listing of people who are not authorized by the blanket policy. Entities on the white-list can easily be traced back to the employee authorizing them by examining their access

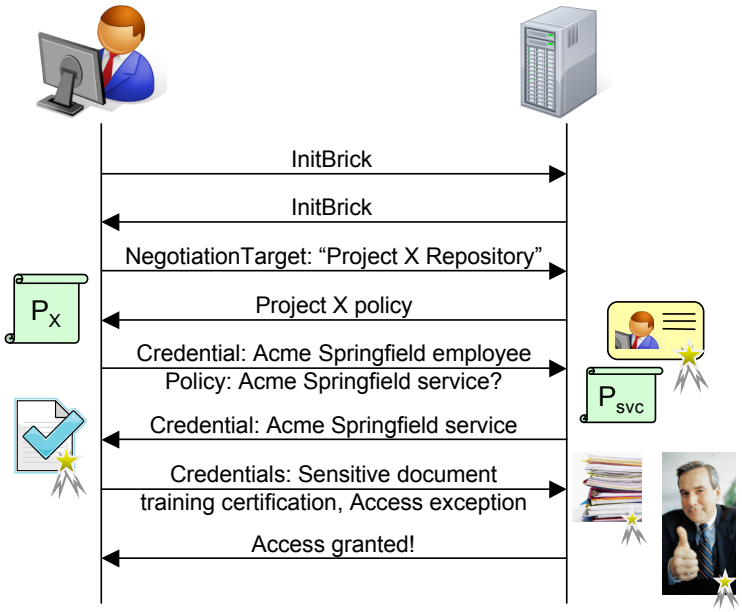

Figure 6.3: A simplified view of the trust negotiation used during our experiments.

exception credential; this is in stark contrast to current ACL-based systems that do not maintain a history of who added whom to the list.

The client in our scenario has a valid employee ID stating that he is a full-time employee in department 2442 of Acme Springfield, a sensitive documents training credential, and an access exception issued by Bob. Figure 6.3 illustrates this example negotiation scenario graphically. The first two messages exchanged during the negotiation contain configuration information used by TrustBuilder2 to establish the parameters for the negotiation session. The second message sent by the client indicates his interest in accessing the file server associated with Project X. The second message sent by the file server releases the policy protecting this file server to the client. The client can satisfy this policy, but is not willing to disclose his security clearance or access exception unless the server can prove that it is operated by Acme Springfield. As such, the third message sent by the client discloses the release policy protecting these credentials and the credential chain ending with his employee ID. Supporting credentials—such as the certificates of Acme Fabrication and Acme Springfield—are not shown in Figure 6.3. At this point, the file server validates the proof-of-ownership associated with the client's employee ID and accepts this credential. It also then discloses the credential chain that certifying that the file service is operated by Acme Springfield. The client verifies this credential chain and the proof-of-ownership associated with the leaf credential in the chain and then discloses his sensitive documents training credential and his access exception to the file server. The file server verifies the proofs-of-ownership

associated with these credentials and then grants the client access to the service.

6.7.2 The Experiments

We used the above trust negotiation scenario to conduct two experiments. The first experiment was designed to enable us to study the average execution time of a trust negotiation session. In this experiment, a client application made a TCP connection to a server application and carried out the trust negotiation described by Figure 6.3 using an ObjectOutputStream to write TrustMessages to the remote server and an ObjectInputStream to read response TrustMessages. When the negotiation succeeded, the client disconnected from the server. This entire process was repeated 100 times. The client and server applications were both executed from the system command prompt using JDK 1.5.0_06.

In the second experiment, we sought to profile the execution of the TrustBuilder2 runtime system to gain a better understanding of the costs of the various components of a trust negotiation. In this experiment, the server process described above was started from the system command line using JDK 1.5.0_06. The client application was loaded into the Eclipse development environment [46] and profiled using the Eclipse Test and Performance Tools Platform (TPTP) tracing and profiling tools plug-in version 4.2.1 [47].

In our experiments, the TrustBuilder2 objects used by the client and server processes supported only the use of X.509 credentials encoded as X509CredentialBrick objects. All X.509 credentials used during this scenario encoded RSA key pairs. Further, each credential was represented as a unique X.509 certificate with its own key pair. Both the client and server processes used the CLOUSEAU compliance checker. The strategy used by both parties was the variant of the TrustBuilder1-relevant strategy discussed in Section 6.5.3 that is implemented by the MaximumRelevantStrategy class included in the TrustBuilder2 distribution. Credential chains were built using the SimpleChainBuilder class and verified using the RootToLeafVerifier class. The IOManipulationModule was disabled at both the client and server. The experiments described above were run using a single machine, rather than two machines, as we were more interested in the computational costs of the trust negotiation than the communication latencies imposed by routing packets through an Ethernet network. The machine that we used had a 3.2 GHz Intel Pentium 4 processor, 1 GB of RAM, and was running Gentoo Linux (kernel 2.6.12).

6.7.3 Results

After conducting the first experiment, we found that the average time to conduct the aforementioned trust negotiation session using TrustBuilder2 was 434.73 ms with a standard deviation of 97.56 ms. This is at least an order of magnitude faster than a trust negotiation session carried out using the original TrustBuilder framework, as a similar negotiation takes seconds on average within that framework [79]. We did find that the first trust negotiation session took roughly three times as long as an average negotiation (1350 ms) due to the cost of Java initially loading the classes used by the TrustBuilder2 framework. We do not see this as a problem, however, as it is likely that TrustBuilder2 objects will be used for multiple negotiations and therefore this initial cost will quickly be amortized, as it was in our experiments.

In our second experiment, we found that the majority of the time was spent in one of three tasks: using the compliance checker ($\approx 49\%$), reading from and writing to I/O streams ($\approx 15.5\%$), and signing proof-of-ownership challenges ($\approx 14.4\%$). What is interesting to note is that the overheads of the remaining portions of the TrustBuilder2 framework amount to less than 0.2% of the overall cost of the trust negotiation process. This implies that the flexibility afforded by the TrustBuilder2 framework does not, in and of itself, carry the steep overheads that we had originally anticipated. Of course, loading inefficient or otherwise expensive plug-ins could easily increase the cost of a trust negotiation.

6.8 Discussion

In this section, we revisit the requirements presented in Section 6.4 and discuss the ways in which they are met by TrustBuilder2. We then discuss potential implications of the performance results obtained in Section 6.7. Lastly, we provide information on obtaining TrustBuilder2.

6.8.1 Requirements Redux

In Section 6.4, we introduced ten requirements that should be provided by frameworks for exploring the systems aspects of trust negotiation. Section 6.6 illustrated the ways in which plug-in extensions to TrustBuilder2 can be used to meet the *arbitrary policy languages*, *arbitrary credential formats*, *interchangeable negotiation strategies*, *flexible policy and credential stores*, and *extensibility* requirements. The plug-in interface for defining strategy modules does not place any constraints on how the

strategy behaves, which enables user-defined strategy modules to meet the *tunable human involvement* and *feature ordering* requirements. The VisualizationModule plug-in to the IOManipulationModule enables advanced logging and visualization features, thus meeting the *advanced logging capabilities* requirement. Finer-grained logging can be accomplished by placing calls to the logger at the mediator hook points described in Section 6.5.2. The QueryModuleMediator can be used to allow the TrustBuilder2 framework to interact with processes external to the negotiation at hand simply by developing new query module plug-ins, thereby meeting the *strategy-driven external interactions* requirement. Finally, each of the plug-ins to the TrustBuilder2 system can be individually enabled or disabled, thereby meeting the *selective feature activation* requirement.

6.8.2 Attacks and Future Research

One striking result from the performance evaluation presented in Section 6.7 is that nearly half of a trust negotiation session is spent interacting with the compliance checker. During our experiments, the client process spent, on average, 226 ms interacting with the compliance checker during a single trust negotiation. This suggests that a novel and highly-effective denial of service attack against trust negotiation-enabled services is to force the use of the remote party's compliance checker. An attacker can easily accomplish this by either placing release policies on every credential that might possibly be released to the remote party, or by sending spurious policies that the remote party *thinks* are protecting resources that could advance the state of the negotiation. Such an attack involves little overhead for the attacker, yet can consume arbitrary amounts of processor time on the host being attacked.

This attack is quite different than the types of denial of service attacks on trust negotiation discussed in the research literature. To date, attacks against trust negotiation systems have focused on examining ways to exploit the credential chain construction and verification processes [81; 104]. These attacks leverage the disparity in cost between transmitting a credential chain and verifying that the chain is correctly formed to consume resources on the target system. The higher per-unit cost of policy compliance checking when compared to credential verification implies that attacking the compliance checker used by a trust negotiation system can be at least as damaging as attacking its credential chain verification process. Furthermore, malicious entities combining these two attacks can slow the processes of analyzing both local and remote policies at the host being attacked.

Analyzing the cost breakdown of example trust negotiation scenarios not only led to the identification of this attack strategy, but also helped identify future research

directions aiming to better optimize trust negotiation systems. For example, an earlier version of our performance analysis led us to explore alternate formulations of the policy compliance checking problem that would allow for more efficient policy analysis than existing theorem proving approaches. The result was the CLOUSEAU compliance checker, which leverages an efficient pattern matching approach to greatly outperform existing compliance checkers both asymptotically and in practice. However, the attacks described above occur even when using this optimized compliance checker. An interesting direction for future research is the development of trust negotiation strategies that can detect the above types of compliance checker abuses and either triage "unproductive" negotiations or seek to limit the use of the compliance checker without compromising the completeness property of the trust negotiation protocol.

6.8.3 Obtaining TrustBuilder2

Since it's initial release in mid-2007, the TrustBuilder2 distribution has been available for download under a BSD-style license at our web site for the project, http://dais.cs.uiuc.edu/tn. The distribution includes the source code and binary distribution of the framework, example client and server applications that make use of TrustBuilder2, programmer documentation (in Javadoc format), and a user manual explaining the installation, configuration, and use of TrustBuilder2. As of July 2008, the TrustBuilder2 framework has been downloaded over 200 times. Compiling the source code for TrustBuilder2 requires the Java Development Kit version 1.5.0 or higher.

6.9 Summary

In this chapter, we presented TrustBuilder2, a flexible framework for investigating the systems aspects of trust negotiation. TrustBuilder2 supports the dynamic loading of new trust negotiation system components—such as strategy modules, compliance checkers, policy and credential storage devices, and logging and visualization modules—without modification to the underlying framework and features an extensible type hierarchy that allows end-users to easily add support for new credential formats and policy languages. We explored several case studies in extensibility, as well as profiled the performance of TrustBuilder2. These studies showed that TrustBuilder2 has a number of desirable properties that make it ideal for researching the systems obstacles to deploying trust negotiation systems in practice. Furthermore, our performance evaluation enabled us to uncover a novel class of

attacks against trust negotiation systems and led us to identify promising areas of future trust negotiation systems research. In the next chapter, we will discuss a novel approach to efficiently solving the policy compliance checking problem that we have developed within this framework.

7 A Pattern Matching Approach to Policy Compliance Checking

Whenever there is a hard job to be done, I assign it to a lazy man; he is sure to find an easy way of doing it.

—Walter Chrysler

In Chapter 6, we showed that checking compliance with trust management policies is one of the most expensive portions of the trust negotiation process. Furthermore, to ensure that a trust negotiation session succeeds whenever possible, authorization policy compliance checkers must be able to find *all* minimal sets of their owners' credentials that can be used to satisfy a given policy. This obviously compounds the inefficiency problem. If, however, all of these satisfying sets can be found *efficiently* prior to choosing which set should be disclosed next, many strategic benefits can also be realized. Unfortunately, solutions to this problem that use existing compliance checkers are too inefficient to be useful in practice. Specifically, the overheads of finding all satisfying sets using existing approaches have been shown to grow exponentially in the size of the union of all satisfying sets of credentials for the policy, even after optimizations have been made to prune the search space for potential satisfying sets [108].

In this chapter, we describe the CLOUSEAU compliance checker. CLOUSEAU leverages an efficient pattern-matching algorithm to find all satisfying sets of credentials for a given policy in time that grows as $O(NA)$, where N is the number of satisfying sets for the policy and A is the average size of each satisfying set. We describe the design and implementation of the CLOUSEAU compliance checker, evaluate its performance as the number and size of satisfying sets for a given policy varies, and show that it vastly outperforms existing approaches to finding all satisfying sets of credentials. To establish the general utility of our approach to compliance checking, we then present a method for automatically compiling policies written in the RT and WS-SecurityPolicy languages into a format suitable for analysis by CLOUSEAU. We further prove the correctness and completeness of these compilation procedures, which ensures that the semantics of the underlying policy languages remain unchanged, despite the use of a radically different analysis approach.[1]

[1]The material presented in this chapter was originally published as [75] and [76].

7.1 Introduction

The design of robust and highly-available trust negotiation systems hinges on the availability of efficient policy *compliance checkers*. Given a policy p and a set C of credentials, the compliance checker is responsible for determining one or more minimal subsets of C that satisfy p. We call these minimal subsets *satisfying sets* of credentials. To ensure that trust negotiation protocols establish trust whenever possible, the compliance checkers used by these systems must be capable of finding all satisfying sets of credentials for a given policy. This enables the negotiator to attempt alternate means of establishing trust in the event that the initially-chosen negotiation tactic leads to a deadlock or other dead end.

Trust negotiation is intrinsically a strategy-driven process in which the participants each attempt to advance the state of the protocol while maximizing their own particular goals [122]. For instance, the so-called *eager* and *parsimonious* negotiation strategies allow negotiation participants to balance a trade-off between negotiation speed and privacy by choosing to disclose either all credentials that a remote party is authorized to view or only those that have been deemed relevant to satisfying the policy at hand, respectively [117]. If a negotiation participant is able to determine *all* satisfying sets of credentials for a given policy a priori, much finer-grained strategies can be employed. For example:

- If entities assign point values to individual credentials that indicate each credential's level of sensitivity (e.g., as in [121]), the negotiation process can respond to a given policy by disclosing the satisfying set with the lowest overall sensitivity.
- In the event that an entity has digital credentials representing memberships in organizations that may lead to various types of discounts or preferential treatment (e.g., AAA, AARP, or frequent flyer credentials), the entity could employ a negotiation strategy that discloses satisfying sets containing these types of credentials first.
- In some cases, entities might wish to minimize the cumulative number of credentials disclosed over multiple rounds of a given trust negotiation session; a simple greedy algorithm could be used to determine the satisfying set that minimizes the overall number of credentials disclosed.
- A party might wish to steer the negotiation in the direction most likely to minimize its duration. For example, a server may wish to lead the negotiation in the direction that the analysis of logs of past negotiations has shown to be

the way that most users gain access.

Existing compliance checkers designed for trust negotiation policy languages find at most one satisfying set of credentials at a time, but can be operated in an iterative manner to discover alternate satisfying sets in the event that the first set found leads to a negotiation path that fails to establish trust. While this iterative approach to discovering satisfying sets is sufficient to ensure the completeness of trust negotiation protocols, it is a very slow way to discover all satisfying sets at once. As a result, it is unrealistic to use this approach as the basis for the types of strategies discussed above. Specifically, the overheads of finding all satisfying sets using such an approach have been shown to grow exponentially in the size of the union of all satisfying sets of credentials for the policy, even after optimizations have been made to prune the search space for potential satisfying sets [108].

In this chapter, we describe the design and implementation of CLOUSEAU, a highly-efficient and policy language-agnostic compliance checker for trust negotiation systems. Rather than discovering satisfying sets of credentials using a top-down proof construction system, CLOUSEAU solves the policy compliance checking problem by compiling policies into a format that can be efficiently analyzed using solutions to the many pattern/many object pattern match problem. Given a set of patterns and a set of objects, algorithms for solving this problem find all patterns matched by subsets of the provided objects. Internally, CLOUSEAU represents access control policies as patterns specifying constraints on the credentials, credential chains, and uncertified claims (e.g., phone numbers, addresses, etc.) that must be presented to gain access to a particular resource. The Rete algorithm [52] is then used to find all satisfying sets by efficiently matching objects representing a user's credentials and claims against these patterns. Overall, CLOUSEAU makes several important contributions related to the compliance checker problem for trust negotiation systems:

- CLOUSEAU requires only tens of milliseconds, on average, to determine *every* satisfying set of credentials associated with a reasonably-sized policy; this is comparable to the time required by previous trust negotiation compliance checkers to find *one* satisfying set.
- To the best of our knowledge, CLOUSEAU represents the first trust negotiation compliance checker capable of finding all satisfying sets of credentials for a given policy with time overheads that scale as $O(NA)$, where N is the number of satisfying sets for a policy and A is the average size of each satisfying set. Previous solutions to this problem have running time overheads that grow exponentially in the size of the union of all satisfying sets. As a concrete point of comparison, the iterative solution presented in [108] takes over 10 seconds

to find two overlapping satisfying sets containing a total of 20 credentials, while CLOUSEAU finds the same satisfying sets in approximately 40 ms.

- In the worst case, the number of satisfying sets for a given policy can be exponential in the size of the policy. However, CLOUSEAU's performance remains reasonable even when policies become inordinately complex. For example, CLOUSEAU can find 512 satisfying sets each of size 9 in approximately one second; we have not found policies of this complexity being used in practice or mentioned elsewhere in the research literature. In Section 7.5, we show that policies as complex as even the most complicated policies used in Becker's formalization of the security requirements for the UK's electronic health records service [16] can be analyzed by CLOUSEAU in under 100 ms.
- Since it can efficiently find all satisfying sets of credentials for a given policy, CLOUSEAU makes the use of "smarter" trust negotiation strategies practical. In Section 7.5, we show that CLOUSEAU is very fast at finding the minimum-weight (e.g., least-sensitive) satisfying set of credentials for a given policy.
- The design of a single highly-optimized compliance checker capable of analyzing policies written in *any* policy language would allow resource owners to write policies without worrying about the costs of analyzing them. CLOUSEAU compiles policies written in high-level policy languages into an intermediate representation that specifies constraints on the actual credentials used to satisfy a given access control policy, which it can then efficiently analyze. We present processes for automatically compiling RT [83] and WS-SecurityPolicy [96] policies into a format that can be analyzed by CLOUSEAU. We then prove the correctness and completeness of these compilation procedures. Since policies written in all existing trust negotiation policy languages are satisfied by the same types of evidence, we expect that equivalent compilation mechanisms could be specified for the other languages as well.

The rest of this chapter is organized as follows. We begin with a discussion of compliance checker modes of operation in Section 7.2. We then formally define the specific instance of the more general policy compliance checking problem solved by CLOUSEAU in Section 7.3. Section 7.4 describes the Rete algorithm, presents the design and implementation of the CLOUSEAU compliance checker, and discusses the internal representation of policies and evidence used by CLOUSEAU. In Section 7.5 we conduct a series of experiments to evaluate the performance of CLOUSEAU and compare its benefits and limitations to those of other compliance checking approaches. We present procedures for automatically compiling RT and WS-SecurityPolicy policies into a format suitable for analysis by CLOUSEAU in Sec-

tions 7.6 and 7.7. Lastly, we summarize the results from this chapter in Section 7.8.

7.2 Types of Compliance Checkers

In [105], the authors broadly classify policy compliance checkers for trust management and trust negotiation systems into three categories. They first define *type-1* compliance checkers as functions that return only a Boolean result indicating whether the policy in question was satisfied. This type of compliance checker is useful in non-iterative systems in which the discovery of *why* a particular access is permitted is superfluous. In this type of system, simply knowing that the compliance checker was able to construct a formal proof of authorization is sufficient. *Type-2* compliance checkers return one satisfying set of credentials in addition to a Boolean value in the case that a policy is found to be satisfied. The ability to associate at least one satisfying set of credentials with a compliance checker decision is *required* by the trust negotiation process. Without such a satisfying set, individuals could not determine which credentials should be sent to their negotiation partner after they determine that a remote policy can be satisfied. Lastly, *type-3* compliance checkers are defined as functions that return every minimal set of credentials that can be used to satisfy a particular policy.

To date, all existing trust negotiation compliance checkers have been developed as type-2 compliance checkers, even though significant strategic benefits could be realized through the use of a type-3 compliance checker. In [108], Smith et al. discuss several important uses of this type of compliance checker and describe the Satisfying Set Generation (SSgen) algorithm for discovering all satisfying sets for a given policy using a type-2 policy compliance checker. They show that when policies are expressed in disjunctive normal form (DNF), then a number of clever optimizations can be made to prune the state space that must be searched for satisfying sets of credentials. They then evaluate the performance of an implementation of the SSgen algorithm that used the IBM TE compliance checker [57] as the base type-2 compliance checker.

Figure 7.1 is a reproduction of the results presented in [108] depicting the running times of the SSGen algorithm. The three most interesting cases in which the SSgen algorithm was evaluated include the cases in which (i) a policy has one satisfying set of size U, (ii) a policy has U satisfying sets of size one, and (iii) a policy has two satisfying sets, each of size $\frac{3U}{4}$. In all cases, U represents the size of the union of all satisfying sets and was varied between 1 and 24. In case (i), the SSgen algorithm scaled linearly with U in the sub-second time range. In cases (ii) and (iii), the SS-

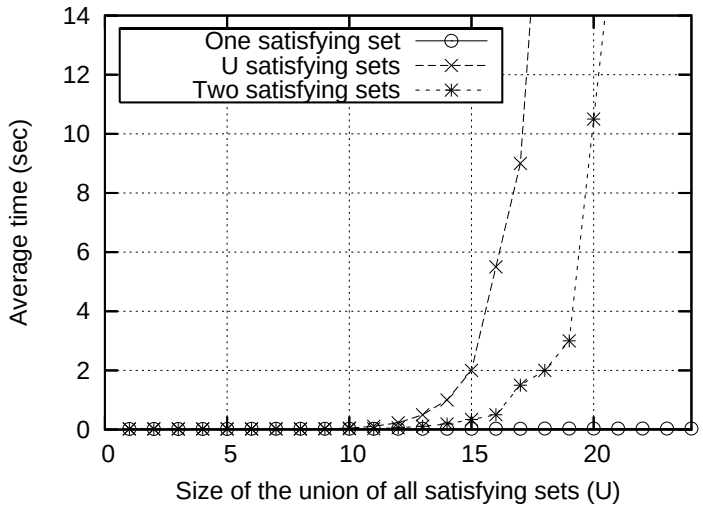

Figure 7.1: Running time of the SSGen algorithm as a function of the size of the union of all satisfying sets.

gen algorithm's running time increased exponentially with U and rapidly became impractical.

As we will see, CLOUSEAU improves upon this previous work by compiling trust negotiation policies into an intermediate representation that can be analyzed using efficient pattern matching algorithms. This non-traditional approach to theorem proving greatly optimizes the process of finding all possible satisfying sets of credentials for a given policy. Further, because existing trust negotiation policy languages can be compiled into a format that can be analyzed using CLOUSEAU, developers can optimize trust negotiation runtime systems while allowing policy writers to continue to use existing high-level policy languages.

7.3 Problem Definition

At its most basic level, a trust negotiation session is a bilateral and iterative exchange of access policies and evidence conducted to establish mutual trust between two parties. For example, a student who wishes to access a resource connected to a computing grid might be returned a policy stating that only full-time students at accredited universities can access that resource. Prior to proving her enrollment status to the resource operator, the student might first require that the resource operator prove that it is operated by either an NSF-sponsored organization or an organization that is a member of the Better Business Bureau. Digital credentials, such as X.509 certificates, are the most common form of evidence used by these proto-

cols, although uncertified claims (as in [20; 24]), proof fragments (as in [12; 119]), or trust tickets (as in [20]) could also be exchanged. In the remainder of this chapter, we will define $\mathcal{E}$ as the set of all such evidence and $\mathcal{P}$ as the set of all policies.

Formally, a compliance checker is defined as a function $cc : 2^{\mathcal{E}} \times \mathcal{P} \rightarrow \mathcal{R}$ that takes some set of evidence and a policy and determines whether (and possibly how) this policy is satisfied by the specified set of evidence. The exact definition of *satisfaction* is specific to the policy language being used. For example, in any language with a model theory, we say that a set E of evidence satisfies a policy p if in every model where E is true, p is also true. In CLOUSEAU, a policy is specified as one or more patterns placing constraints on the credentials and other evidence that must be presented to gain access to a particular resource. We say that such a policy is satisfied if at least one of these patterns can be matched by the set of objects describing the credentials and other evidence possessed by a given entity. As we will see in Section 7.6, proving the correctness of our RT to CLOUSEAU policy compilation process involves proving an equivalence between these two concepts of satisfaction. That is, we must show that an CLOUSEAU pattern-match occurs if and only if the RT rules of inference draw the same conclusion. A similar parallel will be drawn with respect to policies specified using the WS-SecurityPolicy language in Section 7.7.

When a compliance checker is invoked to check the satisfaction of some policy protecting a local resource r, it will be given a set of evidence provided by the remote entity wishing to access r. In this case, the compliance checker need only return a Boolean value indicating whether the policy was satisfied (i.e., $\mathcal{R} \equiv \mathbb{B}$). However, if local credentials are used in an attempt to satisfy a remote policy p, then $\mathcal{R} \equiv 2^{2^{\mathcal{E}}}$. That is, the compliance checker must return zero or more sets of local evidence that minimally satisfy p so that the local entity knows which local evidence can be sent to the remote entity to gain access to the resource protected by p. We say that some set E of evidence *minimally satisfies* a policy p if no proper subset of E also satisfies p. A compliance checker capable of recognizing *all* possible subsets of the local evidence that minimally satisfy a given policy is required to ensure that a trust negotiation protocol will establish trust whenever possible and can also afford its user a number of strategic advantages. Therefore, our focus in this chapter is to efficiently solve the following specific instance of the more general policy compliance checking problem:

THE TYPE-3 COMPLIANCE CHECKER PROBLEM. *Given a set $E \in \mathcal{E}$ of evidence and a policy $p \in \mathcal{P}$, find all distinct subsets sets $e_1, \ldots, e_n$ of E that minimally satisfy p.*

7.4 Design of Clouseau

In this section we discuss the design of the CLOUSEAU compliance checker, which we have designed to efficiently solve the type-3 compliance checker problem. We begin by showing that this problem naturally translates into an instance of the many pattern/many object pattern match problem. This is followed by a discussion of the technical details of our implementation of the CLOUSEAU compliance checker, including an overview of the Rete algorithm, which is used by CLOUSEAU. Lastly, we conclude this section with an overview of CLOUSEAU's internal representation of trust negotiation evidence and policies.

7.4.1 Design Approach

To date, most policy languages for trust negotiation are modeled using logic programming approaches, as the formal semantics of logic programs are well understood. For example, policies in Cassandra [17], PSPL [24], the language used by Koshutanski and Massacci [68], RT [83], and PeerAccess [119] are all specified in this manner. Even some XML-based languages, such as TPL [57] and XACML [66], can be formally modeled using logic programming approaches. Not surprisingly, the policy compliance checking approaches used for these types of languages have leveraged traditional theorem-proving techniques. When the compliance checker is invoked on a policy p with a set E of evidence, the underlying theorem prover stores the set of evidence $e \subseteq E$ used during the construction of a single proof that p was satisfied. Rather than simply returning the Boolean value true, the set e is also returned by the compliance checker to provide support for its decision.

Although this type of theorem-proving approach to the general compliance checker problem is natural given the logical foundations of trust management, it is not the only way in which this problem can be formulated. In fact, using this type of approach to solve the type-3 compliance checker problem is unappealing, as theorem provers in general are designed to find a single proof that a fact is valid (i.e., that a policy is satisfied) and search for alternate proofs only when a given proof attempt fails. As an alternate approach, we have recognized that the the type-3 compliance checker problem is actually an instance of the more general many pattern/many object pattern matching problem [52]:

THE MANY PATTERN/MANY OBJECT PATTERN MATCHING PROBLEM. *Given a set of patterns and a set of objects, determine all of the ways in which the set of objects can be used to match any of the specified patterns.*

Clearly, if credentials and other evidence are treated as objects and policy clauses are treated as patterns, an efficient solution to this problem could likely lead to an efficient solution to the type-3 compliance checker problem. This problem has been studied previously by the artificial intelligence community, as it is central to the design of efficient production system interpreters. As a result, efficient algorithms, such as Rete [52] and TREAT [92], have been developed to solve this problem. The Rete algorithm is optimized for instances of the many pattern/many object pattern matching problem in which (i) patterns are compilable, (ii) all objects remain constant once inserted into the Rete engine's working memory, and (iii) the set of objects changes relatively slowly [52]. Trust negotiation policies are all compilable, as they are designed to enable automated reasoning, rather than human interpretation. Further, credentials and other evidence remain constant once obtained. For example, modifying or tampering with a digital certificate invalidates its attached issuer signature. Lastly, the set of local evidence changes very infrequently and the set of remote evidence grows monotonically as the protocol proceeds. We therefore use the Rete algorithm as the basis for the CLOUSEAU compliance checker.

7.4.2 The Rete Algorithm

We now provide an overview of the Rete algorithm and highlight its benefits in solving the type-3 compliance checker problem. Readers interested in a more in-depth treatment of the Rete algorithm should consult [52] for further information. At a high level, the Rete algorithm works by forming a network of nodes that represent one or more matching tests found in the specified patterns. *Pattern nodes*, which are also known as *one-input nodes*, are used to match single objects stored in the *working memory* of the Rete engine. In the case of CLOUSEAU, these objects represent constraints on individual pieces of trust negotiation evidence such as digital certificates and uncertified claims. The outputs of these pattern nodes can then be fed into one or more *join nodes*, which are used to build more complex patterns consisting of conjunctions of basic patterns and constraints existing between the objects matched by these patterns. In our formulation of the type-3 compliance checker problem, join nodes are used to specify conjunctions of basic credentials as well as inter-credential constraints (i.e., chains of trust or credential delegations).

A collection of pattern nodes and join nodes forms a directed acyclic graph whose sink nodes are called *terminal nodes*. As matches occur in the Rete network, information describing the match is propagated along the edges of the graph. When a terminal node is reached, an event is triggered that signifies that a complete match has occurred. In CLOUSEAU, this implies that a given policy has been satisfied and

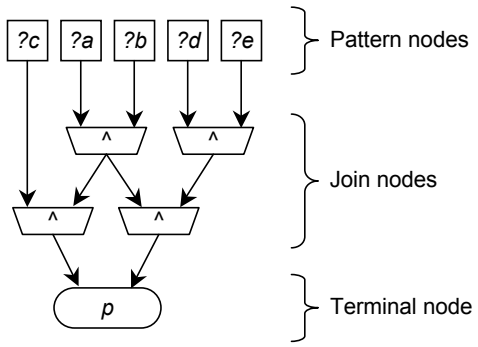

Figure 7.2: An example Rete network.

enables the compliance checker to extract the set of evidence that led to this particular policy satisfaction. Since information is propagated along all possible edges in the Rete network, all satisfying sets are found by the Rete algorithm.

Figure 7.2 is an illustration of a Rete network for the policy $p \leftarrow a \wedge b \wedge (c \vee (d \wedge e))$, which is represented internally as the pair of Horn clauses $p \leftarrow a, b, c$ and $p \leftarrow a, b, d, e$. Square boxes represent pattern nodes, the trapezoids are join nodes, and the rounded rectangle node is a terminal node that represents the satisfaction of the policy p. Note that when using the Rete algorithm, distinct patterns are matched at most once, despite appearing in multiple Horn clauses. Further, join nodes can be shared between multiple clauses of the policy.

Another benefit of the Rete approach is that the network maintains state between invocations, which greatly minimizes the number of times that the working memory is iterated over as multiple policies are matched. That is, each object in the Rete engine's working memory is matched against each pattern node at most one time and the results of this matching operation are saved. For instance, if the policy $p' \leftarrow a \wedge b \wedge (c \vee f)$ is added to the working memory of the Rete engine in Figure 7.2, the Rete engine needs only to check for the existence of credential f, as any matches for $a \wedge b$ and $a \wedge b \wedge c$ were found and memoized during the analysis of the policy p. These types of optimizations further help make the Rete algorithm an efficient approach to solving the type-3 compliance checker problem.

7.4.3 Implementation

We now describe our implementation of CLOUSEAU, a fully-functional compliance checker that leverages the Rete algorithm to efficiently solve the type-3 compliance checker problem. Our goal in designing CLOUSEAU was *not* to propose a new trust negotiation policy language, but rather to explore the design of efficient solu-

tions to the type-3 compliance checker problem. Therefore, rather than designing CLOUSEAU to check the satisfaction of policies specified in one particular policy language (e.g., Cassandra, RT, TPL, or $\mathcal{X}$-TNL), we instead focus on designing a more general-purpose compliance checker. Ultimately, the access control policies used by trust negotiation systems are satisfied by digital certificates or other such evidence presented by participants in the negotiation process. To this end, the policy patterns used to construct the Rete network analyzed by CLOUSEAU specify constraints on the actual evidence (e.g., certificates, certificate chains, and claims) necessary to gain access to a particular resource. This is in contrast to higher-level policy languages, such as RT, which have syntactic constructs to represent concepts such as delegation natively. In Sections 7.6 and 7.7, we discuss processes through which RT and WS-SecurityPolicy policies can be automatically compiled into the native rule format used by CLOUSEAU for analysis. Since all trust negotiation policies are eventually satisfied by the same types of evidence, we believe that equivalent compilation procedures could be derived for other higher-level policy languages as well.

Our implementation of CLOUSEAU was developed using the Java programming language. At a high level, CLOUSEAU takes a set E of evidence and an access control policy p and uses an implementation of the Rete algorithm provided by the Jess expert system [53] to determine all of the ways in which subsets of E can satisfy p. The running time of this Rete implementation scales, on average, linearly with the size of its working memory [53]. This implies that CLOUSEAU's running time scales as $O(NA)$, where N is the number of satisfying sets for a policy and A is the average size of each satisfying set; we confirm this result experimentally in Section 7.5. Our implementation consists of a Jess specification defining the internal representations of evidence and several useful functions for reasoning about credential chains, and a larger Java code base responsible for examining various types of evidence, translating evidence into objects that can be instantiated within CLOUSEAU, and creating and querying the Rete network used by CLOUSEAU. We now discuss the internal representation of evidence used by CLOUSEAU as well as the specification of access control policies.

Evidence Representation

Because Jess provides a general-purpose implementation of the Rete algorithm, it has no way of representing or reasoning about trust negotiation evidence natively. We therefore had to define several *object templates* that represent key types of evidence inside of the Rete engine's working memory. The current implementation of

```
;; Used to describe digital certificates and
;; other credentials
(deftemplate credential
  (slot id)
  (slot issuer)
  (slot subject)
  (slot fingerprint)
  (slot owned (default FALSE))
  (slot map (default (new java.util.HashMap))))

;; Contains an ordered list of credentials making
;; up a credential chain
(deftemplate credential-chain
  (multislot credentials))

;; Claims are stored as attribute/value pairs
(deftemplate claim
  (slot id)
  (slot type)
  (slot value))
```

Figure 7.3: Internal evidence representations used by CLOUSEAU.

CLOUSEAU supports the use of digital certificates, certificate chains, and uncertified claims as forms of evidence; adding support for other types of evidence, such as Trust-$\mathcal{X}$ trust tickets, would be a relatively straightforward process.

CLOUSEAU makes use of TrustBuilder2's extensible credential type hierarchy to allow trust negotiation implementations to add support for new credential types to CLOUSEAU without modifying its underlying code base. Trust negotiation implementations are responsible for validating any proof-of-ownership challenges associated with a given credential and for forming and verifying the credential chains passed into CLOUSEAU. Once a collection of credential chains has been passed into CLOUSEAU, they are translated into instances of the `credential` and `credential-chain` object types described in Figure 7.3. Uncertified claims provided as evidence to CLOUSEAU are represented internally as instances of the `claim` object type.

Objects of type `credential` are generated by extracting information from a given cryptographic credential using methods in the AbstractCredentialBrick class at the top of TrustBuilder2's credential type hierarchy. Each `credential` structure is assigned a unique identifier and contains fields describing the credential's subject and issuer, a cryptographic fingerprint of the credential, a Boolean value indicating whether proof-of-ownership of the credential was verified, and a map containing key/value pairs describing attributes (e.g., job function, hire date, etc.) or other information (e.g., expiry date) embedded in the credential. Internally, credential chains are represented as ordered lists of unique identifiers satisfying two invariants: (i) the credential referenced by the identifier at index 0 in the list is the root

```
;; This policy is satisfied by graduate students at ABET-
;; accredited universities who provide an email address
;; that can be used for future correspondence.
(defrule rule-grad-student
  ;; Find a certificate chain leading from a university
  ;; to a graduate student
  (credential (id ?iuniv) (subject ?suniv))
  (credential (id ?istud) (owned true) (map ?mstud))
  (test (eq "Graduate Student" (?mstud get "Type")))
  (credential-chain (credentials $?cstud))
  (test (is-root ?iuniv ?cstud))
  (test (is-leaf ?istud ?cstud))

  ;; Find a certificate chain leading from ABET to
  ;; the university found above.
  (credential (id ?iabet) (fingerprint
              "38:1A:42:E9:00:7D:19:41:AC:66:F2:EF:12:E6:B4:A1"))
  (credential (id ?icert) (map ?mcert)
              (subject ?scert &: (eq ?scert ?suniv)))
  (test (eq "Accredited University" (?mcert get "Type")))
  (credential-chain (credentials $?ccert)
  (test (is-root ?iabet ?ccert))
  (test (is-nth ?icert 2 ?ccert))

  ;; See if the student provided an email address
  (claim (id ?iemail) (type "Email") (value ?v))
=>
  (assert (satisfaction (resource-name server)
          (credentials ?cstud ?ccert)
          (claims ?iemail))))
```

Figure 7.4: An example CLOUSEAU policy.

of the credential chain and (ii) the credential referenced by the identifier at index $i > 0$ was issued by the owner of the credential at index $i - 1$. Uncertified claims are represented as attribute/value pairs associated with a unique identifier field.

Policy Specification

In CLOUSEAU, access control policies are specified as collections of Jess rules that place constraints on the credentials, credential chains, and uncertified claims that must be presented to gain access to a particular resource. In the remainder of this section, we provide a overview of the CLOUSEAU policy syntax by discussing an example access control policy. We note that only a very small subset of the Jess language is needed to specify CLOUSEAU policies. In particular, we use only the language constructs discussed in this section.

Figure 7.4 is an example access control policy designed to allow graduate students at universities accredited by the Accreditation Board for Engineering and Technology (ABET) to access some resource, provided that they disclose an email address that can be used for future correspondence. Because of the relatively simple na-

ture of this policy, it can be specified using a single Jess rule. Rules consist of two parts: a left hand side (LHS) specifying patterns that must be matched by objects in the working memory of the Rete engine and a right hand side (RHS) that specifies some action to be taken if the pattern in the LHS of the rule is completely matched. These two parts of a rule are separated by the => token.

The LHS of the rule in Figure 7.4 consists of three groups of patterns that must be matched by objects in the working memory of the Rete engine representing trust negotiation evidence. The first group determines whether there exists a certificate chain whose leaf node is a certificate of type "Graduate Student." The first line of this group is a pattern that matches any credential and saves the values of its unique identifier and subject string in the variables `?iuniv` and `?suniv`, respectively. The second line in this grouping is a similar pattern that matches any credential whose ownership was proven during the trust negotiation protocol. The third line in this grouping enforces the constraint that the second credential matched has a "Type" field whose value is "Graduate Student." The last three lines of the first grouping require that the two matched credentials must exist in a credential chain whose authenticity was verified by the trust negotiation implementation. Note that a given pattern need not constrain all fields of the `credential` object type.

The second group of patterns is similar to the first, in that it also establishes the existence of another credential chain. The first two lines of this group form a pattern that matches only the certificate whose cryptographic fingerprint is represented by the hexadecimal string `38:1A:42:E9:00:7D:19:41:AC:66:F2:EF:12:E6:B4:A1`, which is the fingerprint of the (fictitious) certificate used by ABET to issue university accreditations. The third, fourth, and fifth lines of this group form a pattern that matches any credential that has a subject field that is the same as that of the root of the first credential chain (i.e., the university), and has a "Type" field whose value is "Accredited University." The last two lines of this pattern place the constraint that the ABET credential must form the root of a credential chain of length two that ends with the university's accreditation certificate. The last group of constraints consists of a single line specifying that the user needs to also disclose an uncertified claim of type "Email" containing his or her email address, which will presumably be stored for future correspondence.

In general, the RHS of a CLOUSEAU policy can either assert an intermediate result that can be used as input to other rules, or assert a `satisfaction` object describing one way in which a particular policy was satisfied. The former action might be taken if a complicated policy has several paths to satisfaction that each require a common prefix to be matched; we will see examples of this in Section 7.6.2. The policy in Figure 7.4 takes the latter action and asserts a `satisfaction` object con-

taining the set of credentials and the single claim used to satisfy the policy.

Despite a simple policy specification syntax, CLOUSEAU policies can quickly become large and difficult to understand due to the number of constraints that might exist between elements of a credential chain or fields of credentials in different chains. However, we do not view this as a limitation of CLOUSEAU. We do not expect that users of CLOUSEAU will choose to specify policies using this subset of Jess. Rather, we view the native policy representation used by CLOUSEAU as being akin to assembly language in that it provides a representation of a potentially-complex expression that can be efficiently analyzed. We expect that users will specify policies in higher-level trust negotiation policy languages and that these policies will be automatically compiled into CLOUSEAU's native language, much as programs written in high-level programming languages are compiled into assembly code prior to execution. In Sections 7.6 and 7.7, we will make this metaphor more concrete by specifying compilation procedures for policies written using the RT and WS-SecurityPolicy languages.

7.5 Evaluation

In this section, we evaluate the performance of the CLOUSEAU compliance checker and then discuss the implications of our findings. In particular, we examine the amount of time required to find all satisfying sets of evidence for a given policy in three sets of experiments; each set of experiments was conducted on a 2.5GHz Pentium 4 with 512MB RAM running Linux. The running times reported include all overheads associated with generating a Rete network based on the policy rules provided to CLOUSEAU, creating `credential` and `credential-chain` objects corresponding to the credentials and credential chains provided to CLOUSEAU, inserting these objects into CLOUSEAU's working memory, and recovering all satisfying sets of credentials. We first repeat the experiments conducted in [108], which explored the overheads associated with using a type-2 compliance checker to solve the type-3 compliance checker problem.

7.5.1 Experimental Results

The three most interesting cases explored in [108] examined the overheads of using the SSGen algorithm to find all satisfying sets of credentials for a policy in the event that (i) the policy had one satisfying set of size U, (ii) the policy had U satisfying sets of size one, or (iii) the policy had two satisfying sets, each of size $\frac{3U}{4}$. In all

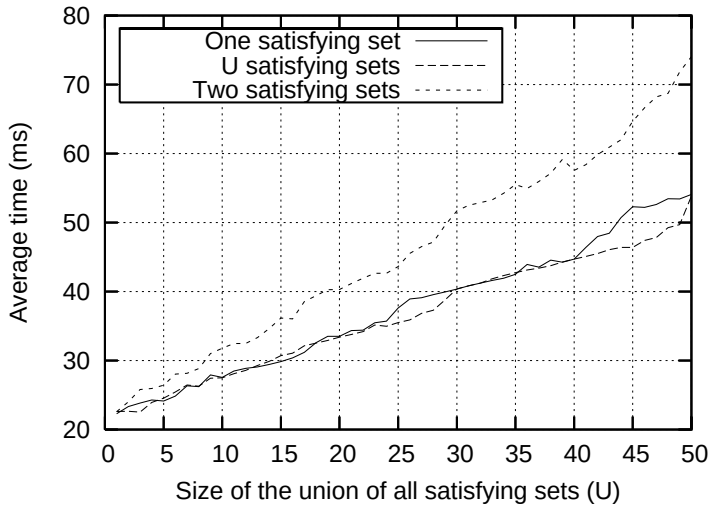

Figure 7.5: Running time as a function of the size of the union of all satisfying sets.

cases, U represents the size of the union of all satisfying sets. Recall from Figure 7.1 in Section 7.2 that the overheads associated with cases (ii) and (iii) grew exponentially in the size of the union of all satisfying sets, and quickly became impractical. Figure 7.5 shows the results of running these same tests using the CLOUSEAU compliance checker; note that the y-axis of Figure 7.1 is labeled in seconds, while the y-axis of Figure 7.5 is labeled in milliseconds. In our experiments, we varied the size of the union of all satisfying sets (U) between 1 and 50. Each data point in the figure represents the average running time over 100 randomly-generated policies; a new Rete network was constructed for each of these 100 trials as to eliminate any optimizations that might occur as a result of partial network reuse (as was discussed in Section 7.4.2). We see that, in all cases, the running time overheads of CLOUSEAU grow linearly with U and never exceed 80 ms to find all satisfying sets. We note also that, unlike the SSgen algorithm, CLOUSEAU does not require that policies be specified in DNF form in order to efficiently find all satisfying sets.

To further explore the running time characteristics of CLOUSEAU, we conducted another experiment designed to more fully examine the types of policies that might be processed by CLOUSEAU in practice. In this experiment, we varied both the number of satisfying sets contained in a particular policy and the size of each satisfying set. For each ⟨number, size⟩ pair, 100 policies were generated at random and examined using CLOUSEAU. The random generation of policies allowed us to explore cases in which satisfying sets overlap with one another to varying degrees. This is important because overlapping satisfying sets will result in shared nodes in the Rete network constructed by CLOUSEAU and, thus, more efficient analysis; examining a random sampling of policies provides us with a more "average case"

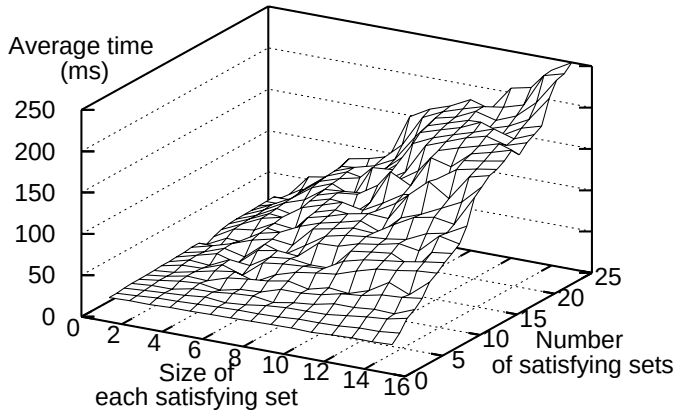

Figure 7.6: Running time as the number of satisfying sets and the size of each satisfying set varies.

view of CLOUSEAU's performance. The results of this experiment are shown in Figure 7.6 and confirm that the performance of CLOUSEAU scales as $O(NA)$, where N is the number of satisfying sets and A is the average size of each satisfying set. We then considered the case in which each credential was assigned a sensitivity value by its owner, as in [121], and ran the above experiments again. Given the satisfying sets detected by CLOUSEAU, it took a trust negotiation strategy, on average, only 0.04 ms to choose the least-sensitive satisfying set to disclose.

We next sought to evaluate the performance of CLOUSEAU in a worst-case scenario. To accomplish this, we analyzed policies of the form $p \leftarrow (c_1 \oplus c_2) \wedge \cdots \wedge (c_{2i-1} \oplus c_{2i})$, which can be satisfied by 2^i different sets of i credentials. Figure 7.7 shows the results of this experiment, which confirm that the performance of CLOUSEAU continues to scale as $O(NA)$ (note that the x-axis of Figure 7.7 follows a logarithmic scale). Policies with exponentially-many satisfying sets are unlikely to be used legitimately in practice, but could be formed by attackers wishing to consume inordinate amounts of system resources. Detecting these types of malicious policies in practice is out of the scope of this thesis, and thus we defer that topic to future work. We do note, however, that CLOUSEAU found 512 satisfying sets of credentials in approximately 1 second. This implies that the naive strategy of capping the time spent in the compliance checker might be a reasonable means of detecting these types of attacks in practice, as finding that many satisfying sets of credentials for a non-attack policy seems exceedingly unlikely.

In all of our experiments, a completely new Rete network was created at each invocation of CLOUSEAU. As stated previously, this was done to eliminate the possible

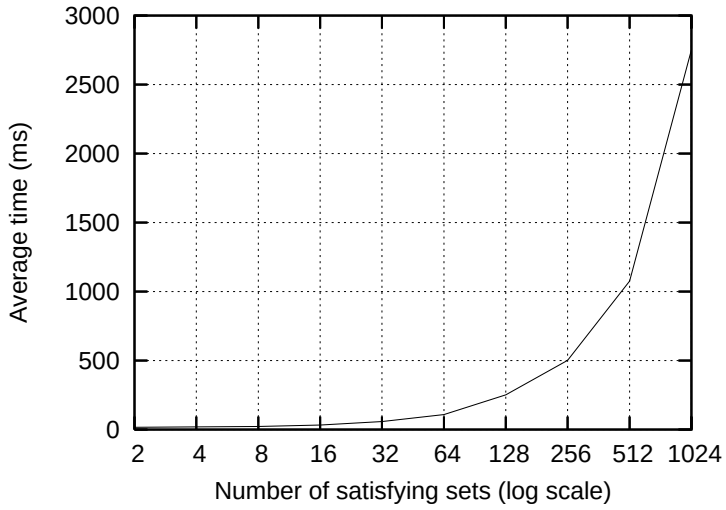

Figure 7.7: Time required to find all 2^i satisfying sets of size i for $i = 1, \ldots, 10$.

benefits of partial network sharing between distinct policies, as network sharing improves the performance of CLOUSEAU. However, some of the primary benefits of the Rete algorithm arise precisely because the state encoded in a particular Rete network can be saved between invocations of the matching algorithm, which reduces the number of times objects (i.e., credentials and other evidence) need to be matched against the pattern nodes (i.e., policy clauses) in the network. If a participant in the trust negotiation process is willing to trade memory for execution time, they could maintain a separate Rete network for each ongoing negotiation constructed using *all* of the policies relevant to that particular negotiation. This would allow them to leverage the statefulness of the Rete algorithm to reduce the number of matching operations required at each invocation of CLOUSEAU and obtain better performance as the negotiation proceeds into later rounds. Further examination of these types of speed versus memory trade offs is an interesting direction for future work.

7.5.2 Discussion

The experiments discussed above illustrate that our CLOUSEAU implementation performs very quickly, requiring only tens of milliseconds to find all satisfying sets of credentials for policies of a reasonable size. Furthermore, these results experimentally confirm our claim that the running time of CLOUSEAU scales as $O(NA)$, where N is the number of satisfying sets for the policy being analyzed and A is the average size of each satisfying set. As always, the number of satisfying sets for a policy p is in the worst case exponential in the size of p, as was the case in the

third set of experiments described above. Even in this case, CLOUSEAU performed admirably, finding 512 satisfying sets of size 9 in just one second.

The largest trust management case study to date is Becker's formalization of the security policies required by the electronic health record service that is being proposed by the United Kingdom's National Health Service [16]. This service, also known as the "Spine," aims to make electronic patient records available to medical personnel, patients, and their designated agents and includes provisions for ensuring the confidentiality of patient records. In [16], Becker completely specifies a collection of Cassandra [17] policies for the NHS Spine and its related services that comply with all available NHS documents describing the requirements for the Spine. The complete Cassandra specification includes definitions of 375 policy rules, 71 roles, and 12 actions that can be taken in the system. Each of the rules is a Horn clause (i.e., a strict conjunction), although often several rules will have the same head. For example, there are several ways in which a clinician can assert that he or she is the "treating clinician" for a particular patient. We will call a set of rules with the same head a policy, since each such set completely specifies the ways in which a user can accomplish a particular goal.

Even the most complex policies specified in [16] contain less than a dozen rules, and thus can be satisfied in at most this many ways. We see from Figures 7.5, 7.6, and 7.7 that policies of this size can be efficiently analyzed by CLOUSEAU in under 100 ms in all cases. This shows that even when the myriad of requirements concerning patient privacy in the medical domain are considered, the number of unique ways that any given policy can be satisfied remains reasonably low. Thus, CLOUSEAU can efficiently handle the largest and most complex set of realistic policies assembled to date.

7.6 Analyzing RT Policies

Although CLOUSEAU is much faster than existing approaches to the policy compliance checking problem, writing trust management policies in the native constraint language analyzed by CLOUSEAU would be a tedious and error-prone process. In the next two sections, we show that policies written in existing policy languages can be compiled into this native representation and analyzed by CLOUSEAU without altering the semantics of the original policy language. The ability to correctly analyze policies written in existing policy languages is a necessary step towards establishing CLOUSEAU as a general-purpose solution to the type-3 compliance checker problem. Furthermore, it allows policy writers to write policies using fa-

miliar abstractions without impacting the costs of analyzing these policies.

In this section, we discuss a method for automatically compiling RT policies into a format suitable for analysis by CLOUSEAU. For ease of exposition, we begin by discussing a compilation process for RT_0 policies, which support the use of unparameterized roles, and prove the correctness and completeness of this process. We then provide an intuition for how this process can be extended to support RT_1 policies allowing the use of parameterized roles. Since other Datalog-based trust negotiation policy languages are very similar to RT, we expect that equivalent compilation procedures could be devised for these languages as well.

7.6.1 RT_0 Policy Syntax

RT_0 is the most basic language in the RT family of trust management languages [83]. As in all of the RT languages, principals are identified by means of identity certificates. RT_0 roles are defined simply as strings identifying the name of the role and cannot be parameterized. Policy statements in RT_0 are expressed as one or more of these role definitions and are encoded as *role definition credentials* signed by the author of the role definition. There are four basic types of role definition credentials in RT_0:

Simple Member A role definition of the form $K_A.R \leftarrow K_D$ encodes the fact that principal K_A considers principal K_D to be a member of the role $K_A.R$.

Simple Containment A role definition of the form $K_A.R \leftarrow K_B.R_1$ encodes the fact that principal K_A defines the role $K_A.R$ to contain all members of the role $K_B.R_1$, which is defined by principal K_B.

Linking Containment A role definition of the form $K_A.R \leftarrow K_A.R_1.R_2$ is called a *linked role*. This defines the members of $K_A.R$ to contain all members of $K_B.R_2$ for each K_B that is a member of $K_A.R_1$.

Intersection Containment The role definition $K_A.R \leftarrow K_{B_1}.R_1 \cap \cdots \cap K_{B_n}.R_n$ defines $K_A.R$ to contain the principals who are members of each role $K_{B_i}.R_i$ where $1 \leq i \leq n$.

These four basic types of role definitions can be used to define a wide range of access control policies. For example, the following RT_0 role definitions express an access control policy requiring that entities accessing a given resource be employees of a SuperGrid member organization:

```
;; Template to store role membership information
(deftemplate is-member
  (slot role)
  (slot roleMgr)
  (slot roleSubj)
  (multislot credentials))

;; Code to detect role memberships via the presence of simple
;; membership policy credentials.  I.e., this can prove that
;; K_A.R <- K_B
(defrule member-of
  ;; Match K_B's identity certificate
  (credential (id ?kb) (fingerprint ?fkb))

  ;; Prove that K_A says that K_B is in role R
  (credential (id ?ka) (fingerprint ?fka))
  (credential (id ?r) (map ?m))
  (test (eq ?fka (?m get "roleMgr")))
  (test (eq ?fkb (?m get "roleSubj")))
  (credential-chain (credentials $?c))
  (test (is-root ?ka ?c))
  (test (is-nth ?r 2 ?c))
=>
  (assert (is-member (role (?m get "role"))
                     (roleMgr (?m get "roleMgr"))
                     (roleSubj (?m get "roleSubj"))
                     (credentials ?ka ?kb ?r))))
```

Figure 7.8: Base policy enabling CLOUSEAU to determine role membership via the use of simple membership credentials.

$$Provider.service \leftarrow Provider.partner.employee$$
$$Provider.partner \leftarrow SuperGrid.memberOrganization$$

If a principal, Alice, could provide credentials proving the statements $SuperGrid.memberOrganization \leftarrow AliceLabs$ and $AliceLabs.employee \leftarrow Alice$, she could satisfy the policy formed by the above two role definitions and gain access to the protected service.

7.6.2 Compiling RT_0 Policies

In RT_0, policies are collections of role definition credentials. Therefore, we must preprocess the set of credentials provided to CLOUSEAU as input in order to generate the set of policy rules that CLOUSEAU will attempt to satisfy. Since CLOUSEAU examines the actual credentials used to hold RT_0 assertions, rather than these higher-level RT_0 assertions, we must make a few assumptions regarding the format of these credentials.

1. We assume that principals in the system are identified by the fingerprint of their identity certificates. Text strings such as "$ABET.accredited$" will be used during the discussion of abstract policies, although such assertions are actually shorthand for statements such as "`38:1A:42:E9:00:7D:19:41:AC:66:F2:EF:12:E6:B4:A1`.*accredited*." When defining CLOUSEAU policies later in this section, we will use the notation `<K_A>` to denote the fingerprint of K_A's identity certificate.

2. Simple membership role definition credentials of form $K_A.R \leftarrow K_B$ are assumed to have the attributes "roleMgr," "role," and "roleSubj" set to the values `<K_A>`, `R`, and `<K_B>`, respectively.

3. Role definition credentials are valid if and only if they are signed by the principal identified at the head of the credential. For example, the simple membership credential $K_A.R \leftarrow K_B$ is considered valid if and only if it is signed by the principal K_A.

Given the above assumptions regarding credential format, we now describe an algorithm for generating a CLOUSEAU policy p' that is equivalent to an RT_0 policy p consisting of the valid role definition credentials $r_1, \ldots, r_n$ and the set of identity certificates $c_1, \ldots, c_m$.

1. Insert the `is-member` template type and the `member-of` rule defined in Figure 7.8 into p'. The `is-member` object type holds information regarding a particular principal's membership in a particular role. The `member-of` rule asserts an `is-member` object if a simple membership role definition credential of form $K_A.R \leftarrow K_B$ can be found, along with identity certificates for K_A and K_B.

2. Generate the `credential` objects corresponding to the identity certificates $c_1, \ldots c_m$ and insert these into the working memory of CLOUSEAU.

3. For each valid role definition credential r_i:

   - Generate the `credential` object corresponding to r_i and insert it into the working memory of CLOUSEAU. Save the "id" field of this object as the variable `<id>`.

   - If r_i is a simple containment credential of form $K_A.R \leftarrow K_B.R_1$ then insert the following rule into p':

```
(defrule rule-sc-<id>
  (is-member (role "R_1") (roleMgr <K_B>)
             (roleSubj ?rs) (credentials $?c))
=>
  (assert (is-member (role "R") (roleMgr <K_A>)
```

```
                    (roleSubj ?rs)
                    (credentials ?c <id>)))
```

This rule asserts that a principal is a member of role $K_A.R$ if he is also a member of $K_B.R_1$.

- If r_i is a linking containment credential of form $K_A.R \leftarrow K_A.R_1.R_2$ then insert the following rule into p':

```
(defrule rule-lc-<id>
  ;; Find a member of R_2
  (is-member (role "R_2") (roleMgr ?r2mgr)
             (roleSubj ?r2subj) (credentials $?cr2))
  ;; find a member of K_A.R_1
  (is-member (role "R_1") (roleMgr <K_A>)
             (roleSubj ?r1subj) (credentials $?cr1))
  (test (eq ?r1subj ?r2mgr))
=>
  (assert (is-member (role "R") (roleMgr <K_A>)
                     (roleSubj ?r2subj)
                     (credentials ?cr1 ?cr2 <id>)))
```

This rule asserts that a principal is a member of the role $K_A.R$ if he is a member of the role R_2 defined by some member of $K_A.R_1$.

- If r_i is an intersection containment credential of form $K_A.R \leftarrow K_{B_1}.R_1 \cap \cdots \cap K_{B_k}.R_k$ then insert the following rule into p':

```
(defrule rule-ic-<id>
  (is-member (role "R_1") (roleMgr <K_B_1>)
             (roleSubj ?rs1) (credentials $?cr1))
  ...
  (is-member (role "R_k") (roleMgr <K_B_k>)
             (roleSubj ?rsk &: (eq ?rs1 ?rsk))
             (credentials $?crk))
=>
  (assert (is-member (role "R") (roleMgr <K_A>)
                     (roleSubj ?rs1)
                     (credentials ?cr1 ... ?crk <id>)))
```

This rule asserts that a principal is a member of the role $K_A.R$ if role memberships for each of the roles $K_{B_1}.R_1, \ldots, K_{B_k}.R_k$ can be found. Note that this rule enforces the constraint that these role membership credentials must all refer to the same subject principal.

4. Given the target role for the negotiation, say $K_A.R_t$, insert the following rule into p':

```
(defrule target
  (is-member (role "R_t") (roleMgr <K_A>) (roleSubj ?rs) (credentials $?c))
  (credential (fingerprint ?fp &: (eq ?fp ?rs)) (owned TRUE))
=>
  (assert (satisfaction (resource-name "target") (credentials ?c))))
```

This rule triggers the insertion of a policy satisfaction object whenever an identity certificate with valid proof-of-ownership can be found for a member of the target role $K_A.R_t$.

Intuitively, this compilation process works in a bottom-up fashion, as follows. The `member-of` rule enables CLOUSEAU to conclude that a principal K_B is a member of the role $K_A.R$ if it finds K_B's identity certificate, K_A's identity certificate, and the simple membership role definition certificate declaring K_B to be a member of $K_A.R$. These three credentials are retained as evidence supporting K_B's membership in $K_A.R$. The rules inserted during step 3 of the compilation process can then combine these basic role membership assertions to prove membership in roles defined by more complex expressions (i.e., simple containment, linking containment, and intersection containment).

As role membership assertions are combined by these rules, the credential identifiers stored in these role membership assertions are combined and stored in the newly-concluded role membership assertions. Finally, the `target` rule inserted into p' at step 4 of the compilation process asserts a `satisfaction` object whenever membership in the target role of the negotiation process can be found for a principal who could demonstrate proof-of-ownership of his or her identity certificate. This allows CLOUSEAU to conclude that the policy in question has been satisfied and to extract the credentials used during the satisfaction process. The Rete algorithm ensures that all paths leading to the creation of a `satisfaction` object are explored during the pattern matching process, which implies that all satisfying sets of evidence are discovered by the CLOUSEAU compliance checker. Further, an increase in the size of the RT policy to be analyzed causes only a linear increase in the running time of CLOUSEAU. This is a result of the fact that the size of CLOUSEAU's working memory is increased linearly for each credential analyzed during the policy compilation process (i.e., we add at most one rule to CLOUSEAU for each credential processed).

Figure 7.9 illustrates the result of applying the above compilation process to the RT_0 access policy described in Section 7.6.1. We now make the following claim regarding the correctness and completeness of this policy compilation process.

THEOREM 7.6.1 (Correctness and Completeness). *Let $R = \{r_1, \ldots, r_n\}$ be a set of role definition credentials, $C = \{c_1, \ldots, c_m\}$ be a set of identity certificates, $p = R \cup C$ be an RT_0 policy, and let p' be the result of compiling p using the above process.* CLOUSEAU *finds the satisfying set $S \subseteq (R \cup C)$ of credentials for the policy p' if and only if the RT rules of inference can be used on exactly the set S of credentials to prove membership in the target role.*

```
;; Template to store role membership information
(deftemplate is-member
  (slot role)
  (slot roleMgr)
  (slot roleSubj)
  (multislot credentials))

;; Code to detect role memberships via the presence of simple
;; membership policy credentials.  I.e., this can prove that
;; K_A.R <- K_B
(defrule member-of
  ;; Match K_B's identity certificate
  (credential (id ?kb) (fingerprint ?fkb))

  ;; Prove that K_A says that K_B is in role R
  (credential (id ?ka) (fingerprint ?fka))
  (credential (id ?r) (map ?m))
  (test (eq ?fka (?m get "roleMgr")))
  (test (eq ?fkb (?m get "roleSubj")))
  (credential-chain (credentials $?c))
  (test (is-root ?ka ?c))
  (test (is-nth ?r 2 ?c))
=>
  (assert (is-member (role (?m get "role"))
                     (roleMgr (?m get "roleMgr"))
                     (roleSubj (?m get "roleSubj"))
                     (credentials ?ka ?kb ?r))))

;; Provider.service <- Provider.partner.employee
(defrule rule-1
  (is-member (role "employee") (roleMgr ?r2mgr)
             (roleSubj ?r2subj) (credentials $?cr2))
  (is-member (role "partner") (roleMgr <Provider>)
             (roleSubj ?r1subj)
             (credentials $?cr1))
  (test (eq ?r1subj ?r2mgr))
=>
  (assert (is-member (role "service") (roleMgr <Provider>)
                     (roleSubj ?r2subj)
                     (credentials ?cr1 ?cr2 <id>)))

;; Provider.partner <- SuperGrid.memberOrganization
(defrule rule-2
  (is-member (role "memberOrganization") (roleMgr <SuperGrid>)
             (roleSubj ?rs) (credentials $?c))
=>
  (assert (is-member (role "partner") (roleMgr <Provider>)
                     (roleSubj ?rs) (credentials ?c <id>)))

;; Provider.service is our target role
(defrule target
  (is-member (role "service") (roleMgr <Provider>)
             (roleSubj ?rs) (credentials $?c))
  (credential (fingerprint ?fp &: (eq ?fp ?rs))
              (owned TRUE))
=>
  (assert (satisfaction (resource-name "target")
                        (credentials ?c))))
```

Figure 7.9: A compiled version of the RT_0 policy discussed in Section 7.6.1.

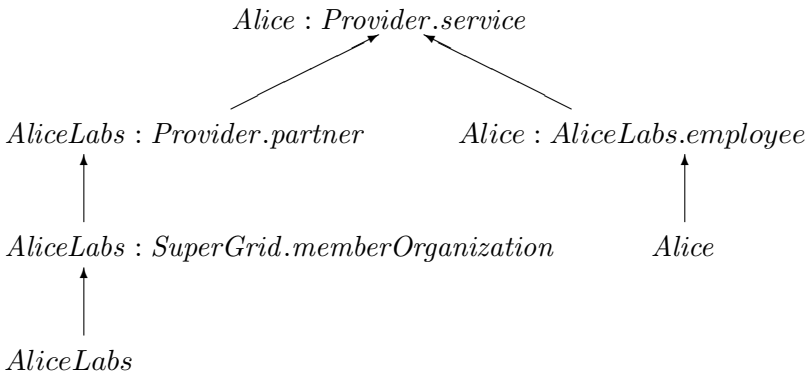

Figure 7.10: An example RT_0 proof tree.

We now present the full proof of the above theorem. Readers not wishing to examine the details of this proof can skip to Section 7.6.4.

7.6.3 Proof of Theorem 7.6.1

A role membership proven using the RT rules of inference can be represented as a proof tree in which the root node represents the target role (i.e., $K_A.R$), intermediate nodes represent memberships in intermediate roles, and all leaves of the tree represent identity certificates. All non-leaf nodes of the proof tree are labeled with the identity of the principal whose membership in that particular role has been proven. Figure 7.10 is an example proof tree proving that Alice is a member of the $Provider.service$ role as discussed in Section 7.6.1. The linking containment rule $Provider.service \leftarrow Provider.partner.employee$ causes the proof tree to branch: the left branch proves that AliceLabs is a a member of the $Provider.partner$ role and the right branch proves that Alice is an employee of AliceLabs. Given this representation of a role membership verified using the RT rules of inference, we now prove the following.

LEMMA 7.6.2. *Let $R = \{r_1, \ldots, r_n\}$ be a set of role definition credentials, $C = \{c_1, \ldots, c_m\}$ be a set of identity certificates, $p = R \cup C$ be an RT_0 policy, and let p' be the result of compiling p using the process described in Section 7.6.2. If a proof tree with root node $K_C : K_A.R$ can be found for p using the RT rules of inference on a set $S \subseteq (R \cup C)$ of credentials, then* CLOUSEAU *finds the satisfying set S of credentials proving membership in $K_A.R$ after analyzing the compiled policy p'.*

Proof. We proceed by induction on the depth of the proof tree found using the

RT rules of inference. The base case occurs when the proof tree extends one level beyond the root node. In this case, the proof tree is the result of a simple membership role definition $K_A.R \leftarrow K_C$. The root node of this proof tree is labeled $K_C : K_A.R$ and the leaf node of the tree signifies that K_C's identity certificate was obtained. The CLOUSEAU compliance checker determines that the policy p' generated by compiling this base policy was satisfied, as follows. The simple membership credential containing the assertion $K_A.R \leftarrow K_C$, K_A's identity certificate, and K_C's identity certificate match the LHS of the `member-of` rule inserted during step 1 of the policy compilation process (see Figure 7.8). This rule asserts an `is-member` object defining K_C to be a member of $K_A.R$ and stores references to the simple membership credential, as well as the identity certificates of K_C and K_A.

Assume that CLOUSEAU can determine membership in any role using the same set of credentials as RT in all cases where the depth of the proof tree found using the RT rules of inference is at most n. To prove that CLOUSEAU can determine role membership using the same set of credentials as RT where the proof tree found using the RT rules of inference is of depth $n + 1$, we must consider three cases.

Case 1 (Simple Containment): In this case, the RT inference linking the first and second levels of the proof tree will occur as a result of processing some simple containment credential $K_A.R \leftarrow K_B.R_1$. The root of the resulting proof tree has one child node whose label is $K_C : K_B.R_1$. The subtree of the proof rooted at this child node is of depth n and thus CLOUSEAU finds the same set of credentials proving K_C's membership in $K_B.R_1$ (by the inductive hypothesis) and will assert an `is-member` object containing this information. This will match the LHS of the `rule-sc-<id>` rule inserted after preprocessing the $K_A.R \leftarrow K_B.R_1$ simple containment credential during step 3 of the compilation process, which will then insert an `is-member` object defining K_C to be a member of $K_A.R$.

Case 2 (Linking Containment): In this case, the RT inference linking the first and second levels of the proof tree will occur as a result of processing a linking containment credential $K_A \leftarrow K_A.R_1.R_2$. The root of the resulting proof tree will have two child nodes: one labeled $K_B : K_A.R_1$ asserting that some principal K_B is a member of the role $K_A.R_1$, and one labeled $K_C : K_B.R_2$, which asserts that K_C is a member of $K_B.R_2$. Because the subproofs rooted at these two nodes each have a depth of at most n, CLOUSEAU will have asserted `is-member` objects describing these memberships using the same set of credentials, by the inductive hypothesis. These two objects will then match the LHS of the `rule-lc-<id>` rule inserted after preprocessing the $K_A \leftarrow K_A.R_1.R_2$ linking containment credential during step 3 of the compilation process, which will assert K_C's membership in $K_A.R$.

Case 3 (Intersection Containment): In this case, the RT inference linking the first and second levels of the proof tree will occur as a result of processing an intersection containment credential $K_A.R \leftarrow K_{B_1}.R_1 \cap \cdots \cap K_{B_m}.R_m$. The root of the resulting proof tree will have m child nodes, each labeled to assert K_C's membership in some role $K_{B_i}.R_i$. Because the subproofs rooted at these m nodes are of depth at most n, CLOUSEAU can assert `is-member` objects providing evidence of K_C's membership in these roles using the same sets of credentials, by the inductive hypothesis. These m objects will then match the LHS of the `rule-ic-<id>` rule inserted after preprocessing the $K_A.R \leftarrow K_{B_1}.R_1 \cap \cdots \cap K_{B_m}.R_m$ intersection containment credential during step 3 of the compilation process, which will assert K_C's membership in $K_A.R$.

Since CLOUSEAU can discover the same satisfying set of credentials as RT in each of these three cases, it can do so for any policy p' resulting from the compilation of an RT_0 policy p. For RT to grant access based on this proof, however, K_C must demonstrate proof of ownership of his or her identity certificate. CLOUSEAU enforces this same constraint by means of the `target` rule, which also requires a demonstration of proof-of-ownership for K_C's identity certificate. □

LEMMA 7.6.3. *Let $R = \{r_1, \ldots, r_n\}$ be a set of role definition credentials, $C = \{c_1, \ldots, c_m\}$ be a set of identity certificates, $p = R \cup C$ be an RT_0 policy, and let p' be the result of compiling p using the process described in Section 7.6.2. If* CLOUSEAU *asserts a* `satisfaction` *object containing the set $S \subseteq (R \cup C)$ of credentials that proves K_C's membership in the target role $K_A.R$, then this membership can also be proven using the RT rules of inference on S.*

Proof. We say that a set of CLOUSEAU rules $l_1, \ldots, l_j$ forms a *chain* if an assertion made by the RHS of l_1 matches a pattern on the LHS of l_2, an assertion made by the RHS of l_2 matches a pattern on the LHS of l_3, and so on. Our proof then proceeds by induction on the length of the longest chain of rules invoked by CLOUSEAU prior to invoking the `target` rule. The base case occurs when one rule is invoked. This can only occur when the `member-of` rule is matched by a simple membership credential $K_A.R \leftarrow K_C$, the identity certificates of K_A and K_C. This will assert an `is-member` object attesting to K_C's membership in $K_A.R$. In this case, we can use the RT rules of inference to determine that K_C is a member of $K_A.R$, as we have both the simple membership credential $K_A.R \leftarrow K_B$ and K_C's identity certificate.

Now, assume that the RT rules of inference can be used on the policy p to determine membership in $K_A.R$ as long as the length of the longest chain of rules invoked by CLOUSEAU when analyzing the policy p' is less than or equal to n. To prove that the RT rules of inference can be used to determine membership in $K_A.R$ if the longest

chain of rules invoked by CLOUSEAU is $n + 1$, we must examine three cases.

Case 1 (`rule-sc-<id>`): Consider the case in which the last rule in a chain to be invoked by CLOUSEAU is an instance of the `rule-sc-<id>` inserted by CLOUSEAU upon examining a simple containment credential $K_A.R \leftarrow K_B.R_1$. The LHS of this rule would be matched by a single `is-member` object asserting K_C's membership in the role $K_B.R_1$. Since the longest chain of CLOUSEAU rules needed to assert this object is at most $n - 1$, the RT rules of inference could also be used to prove membership in $K_B.R_1$ using the same set of credentials, by the inductive hypothesis. This membership proof can be combined with the simple containment credential $K_A.R \leftarrow K_B.R_1$ to prove K_C's membership in $K_A.R$ using the RT rules of inference.

Case 2 (`rule-lc-<id>`): Consider the case in which the last rule in a chain to be invoked by CLOUSEAU is an instance of the `rule-lc-<id>` inserted by CLOUSEAU upon examining a linking containment credential $K_A.R \leftarrow K_A.R_1.R_2$. The LHS of this rule would be matched by two `is-member` objects: one proving that some K_B is a member of $K_A.R_1$ and another proving that K_C is a member of $K_B.R_2$. The longest chain of CLOUSEAU rules needed to assert either of these objects is at most $n - 1$ and thus the RT rules of inference could also be used to prove $K_A.R_1 \leftarrow K_B$ and $K_B.R_2 \leftarrow K_C$ using the same set of credentials. These membership proofs can be combined with the linking containment credential $K_A.R \leftarrow K_A.R_1.R_2$ to prove K_C's membership in $K_A.R$ using the RT rules of inference.

Case 3 (`rule-ic-<id>`): Consider the case in which the last rule in a chain to be invoked by CLOUSEAU is an instance of the `rule-ic-<id>` inserted by CLOUSEAU upon examining an intersection containment credential $K_A.R \leftarrow K_{B_1}.R_1 \cap \cdots \cap K_{B_m}.R_m$. The LHS of this rule would be matched by m `is-member` objects each proving that K_C is a member of some $K_{B_i}.R_i$. The longest chain of CLOUSEAU rules needed to assert any of these objects is at most $n - 1$ and thus the RT rules of inference could also be used to prove $K_{B_i}.R_i \leftarrow K_C$ for $1 \leq i \leq m$ using the same set of credentials. These memberships can be combined with the intersection containment credential $K_A.R \leftarrow K_{B_1}.R_1 \cap \cdots \cap K_{B_m}.R_m$ to prove K_C's membership in $K_A.R$ using the RT rules of inference.

Since the RT rules of inference can be used to find an equivalent proof of membership in the role $K_A.R$ for each of these three cases, we conclude that the RT rules of inference can be used on the policy p to determine an equivalent proof of membership for the principal K_C in the role $K_A.R$ any time that CLOUSEAU asserts an `is-member` object when analyzing the policy p'. For CLOUSEAU to grant access based on this role membership (i.e., assert a `satisfaction` object), proof-

of-ownership of K_C's identity certificate is required by the `target` rule. RT also requires this proof-of-ownership prior to granting access, as the label of the root node of the proof tree is K_C. □

Theorem 7.6.1 follows directly from Lemmas 7.6.2 and 7.6.3.

7.6.4 Supporting RT_1 Policies

The only feature that RT_1 adds to RT_0 is the ability to parameterize role definitions. For example, rather than requiring Alice's employee credential to be of the form $AliceLabs.employee \leftarrow Alice$, it could instead encode other attributes regarding Alice's employment. For example, we could define Alice as the President of AliceLabs by defining the following simple membership credential:

$$AliceLabs.employee(title=\text{“}President\text{”}) \leftarrow Alice$$

RT_1 role definition credentials can also constrain role memberships based on the values of role parameters. For example, the following simple containment credential declares that only widgets whose price is over $10 are on sale:

$$Acme.sale \leftarrow Acme.widget(price > 10)$$

Adding support for the above types of parameterizations and constraints to the policy compilation process described in Section 7.6.2 is a relatively straightforward process. In fact, we must only (i) provide support for storing parameters and their values in simple membership role definition credentials and (ii) allow the various containment role definition rules generated during the policy compilation process to place constraints on these parameter values. To address (i), we can store the parameters of a given simple membership credential and their corresponding values in the "map" field of the simple membership's CLOUSEAU `credential` object. Further, the `is-member` object template must be extended to include this mapping of parameter names to values.

Addressing point (ii) is a slightly more complicated process involving the generation of CLOUSEAU rules during step 3 of the compilation process. Rather than explain each case in detail, we will instead provide one example compilation rule and claim that the other cases can be handled in a similar fashion. As an example, consider the following simple containment role definition credential:

$$AliceLabs.seniorEmployee \leftarrow AliceLabs.employee(hireYr < 1995)$$

This credential defines members of the "Senior Employee" role at AliceLabs to contain all employees hired before the year 1995. A policy compiler could parse the above type of role definition credential to form the following rule:

```
(defrule rule-<id>
  (is-member (role "employee") (roleMgr <AliceLabs>) (map ?m)
             (roleSubj ?rs) (credentials $?c))
  (test (> 1995 (?m get "hireYr")))
=>
  (assert (is-member (role "seniorEmployee") (roleSubj ?rs)
                     (roleMgr <AliceLabs>) (credentials ?c <id>)))
```

Creating this rule automatically involves an extension to the rule generation logic presented in Section 7.6.2 that adds one `test` clause to the rule for every constraint placed on an attribute value by the RT_1 role definition credential. Since the Rete engine used by CLOUSEAU supports comparison operators such as $>$, $<$, and $=$, it can support the types of constraints allowed by RT_1. As such, it is possible to define an automated procedure for compiling RT_1 policies into a format suitable for analysis by CLOUSEAU.

7.7 Analyzing WS-SecurityPolicy Policies

In addition to efficiently analyzing Datalog-based policy languages, CLOUSEAU can also be used to check compliance with policies specified using less formal, industry-standard policy languages. We now describe how trust management policies can be specified within the framework provided by the WS-SecurityPolicy [96] standard. We then show that these policies can also be compiled into a format suitable for analysis by CLOUSEAU, as well as prove that the compilation procedure that we specify is both correct and complete.

7.7.1 Basic Policy Syntax

WS-SecurityPolicy takes a token-based approach to security, in that policies identify specific *security tokens* that must be presented in order to gain access to a particular service. The WS-SecurityPolicy specification defines policy assertions that can be used to require the use of Kerberos tickets, SAML assertions, and X.509 certificates, as well as other security token formats. As an example, the following policy assertion requires the use of an X.509 certificate issued by the Better Business Bureau's (fictitious) security token service:

```
<sp:X509Token xmlns:sp="..." xmlns:wsa="...">
  <sp:IssuerName>C=US/O=Better Business Bureau/CN=sts.bbb.org</sp:IssuerName>
</sp:X509Token>
```

Trust negotiation policies are typically more complicated than this, however, as they can include *multiple* attribute constraints. Requiring the use of multiple security tokens can be accomplished through the use of the basic policy connectives defined by WS-Policy [48]. WS-Policy defines the `ExactlyOne` and `All` connectives, which require that either *one* or *all* subclauses of a particular clause in a policy be satisfied in order for that clause of the policy to be satisfied. Although these two connectives can be used to express any arbitrary policy structure, the WS-Policy specification recommends that policies be expressed in disjunctive normal form (DNF) using a policy structure of the following form:

```
<wsp:Policy xmlns:wsp="...">
  <wsp:ExactlyOne>
    <wsp:All>
      <!-- Policy assertion (1,1) -->
      ...
      <!-- Policy assertion (1,n_1) -->
    </wsp:All>
    ...
    <wsp:All>
      <!-- Policy assertion (m,1) -->
      ...
      <!-- Policy assertion (m,n_m) -->
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

The `ExactlyOne` clause in the above policy indicates that exactly one of its child nodes must be satisfied in order for the policy to be satisfied. Since each child node is an `All` clause, satisfying this policy requires that *all* policy assertions (`i`, `j`) are satisfied for some `i` where $1 \leq \texttt{i} \leq \texttt{m}$ and for all `j` such that $1 \leq \texttt{j} \leq \texttt{n\_i}$.

Combining the security token policy assertions from WS-SecurityPolicy with the policy connectives defined in WS-Policy allows us to specify a range of interesting trust negotiation policies. For example, consider a service that wishes to be protected by a policy requiring that users present X.509 certificates issued by the registrar of State University and the ACM; this would indicate that authorized users of the service need to be students of State University and members of the ACM. Assuming that State University's registrar and the ACM each run an online security token service (STS) that manages the credentials issued within their respective domains, Figure 7.11 illustrates how such a policy could be written in a standards-compliant manner.

```
<wsp:Policy xmlns:wsp="..." xmlns:sp="...">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:X509Token xmlns:wsa="...">
        <sp:IssuerName>
          C=US/O=State University/OU=Registrar/CN=sts-reg.stateu.edu
        </sp:IssuerName>
      </sp:X509Token>
      <sp:X509Token xmlns:wsa="...">
        <sp:IssuerName>
          C=US/O=ACM/CN=sts.acm.org
        </sp:IssuerName>
      </sp:X509Token>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Figure 7.11: An example trust negotiation policy requiring users to present X.509 certificates from the State University registrar, as well as the ACM.

7.7.2 Encoding Advanced Attribute Constraints

While the above, strictly token-based approach to trust negotiation policy specification works in some circumstances, it is inadequate for others. For example, consider complex credentials such as driver's licenses that contain information about the type of vehicles the bearer is authorized to drive and the date of birth of the bearer, or employee IDs that indicate the employee's rank, department, and year of hire. The policies used in the previous section can only determine whether an entity has an employee ID or driver's license, but cannot constrain the attribute fields—also known as claims—encoded in the certificate.

The authors of the WS-SecurityPolicy and WS-Trust [97] standards recognize that placing constraints on the claims encoded in a security token is an important aspect of security policy specification. As such, these standards define an optional `Claims` element that can be included in the security token policy assertions that make up a given security policy. These standards do not specify the fields comprising a given `Claims` element; to allow for maximum extensibility, third parties can define claims *dialects* that specify the format and contents of these elements.

To facilitate the use of more expressive—yet standards-compliant—trust negotiation policies within the WS-SecurityPolicy framework, we have developed one such claims dialect. Our claims dialect allows policy writers to place an arbitrary number of (attribute name, comparison operator, value) constraint triples on the claims encoded in a security token. This format was chosen because it is sufficiently expressive to represent instances of the constraint checking problem. For example, the constraint triple (License Type, EQ, CDL) would require that the "Li-

Element	Description
/cl:Claim	This element is used to encode a constraint on a claim encoded in the security token to which it refers. These constraints take the form of (attribute, operation, value) triples.
/cl:Claim/cl:Attribute	The name of the attribute or claim to which this constraint refers.
/cl:Claim/cl:Op	The operation portion of a constraint triple. Acceptable values for this field are `EQ`, `GT`, `LT`, `GTEQ`, and `LTEQ`. These values denote "equals," "greater than," "less than," "greater than or equal to," and "less than or equal to," respectively.
/cl:Claim/cl:Value	The value field of the constraint triple.
/cl:Ownership	This element is used to indicate whether proof of ownership of the security token to which it refers needs to be demonstrated when the token is disclosed.
/cl:Ownership/@Status	This optional attribute may be set to either `true` or `false` depending on whether proof of ownership is required. If this attribute is not present, a default value of `true` is assumed.

Table 7.1: Descriptions of the elements making up our claims dialect.

cense Type" field of a particular driver's license security token be set to the value "CDL." Furthermore, our claims dialect provides a mechanism through which policy writers can require not only the disclosure of a particular security token, but also a demonstration of proof-of-ownership. This enables explicit differentiation between credentials that must be *owned* by the individual requesting access to a particular service and other supporting credentials that must be presented. The XML elements defined by this claims dialect are summarized in Table 7.1; a more detailed treatment of this claims dialect can be found in the XML schema defining the dialect (see Figure 7.12).

Figure 7.13 contains a more complex version of the policy presented in Figure 7.11. This version of the policy leverages our claims dialect to restrict service access to *graduate* students of State University who have been members of the ACM *since at least 2006*. The use of the `Ownership` element inside each of the `Claims` elements requires that proof of ownership be demonstrated for both tokens.

7.7.3 Compiling WS-SecurityPolicy Policies

We now describe a compilation procedure that can be used to translate trust management policies specified using WS-SecurityPolicy into a format that is suitable for analysis by CLOUSEAU. This translation is actually quite natural, as the token-based approach to trust and security embodied by WS-Trust and its related standards maps directly onto the credential-based intermediate policy language used by CLOUSEAU.

In presenting the following compilation procedure, we assume that policies are expressed in DNF, as recommended by [48]. That is, we assume that policies are a collection of n `All` clauses, each identifying one satisfying set of security tokens

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://dais.cs.uiuc.edu/claim.xsd"
           xmlns="http://dais.cs.uiuc.edu/claim.xsd"
           elementFormDefault="qualified">

  <xs:element name="Claim">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Attribute" type="xs:string"/>
        <xs:element name="Op">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="LT"/>
              <xs:enumeration value="GT"/>
              <xs:enumeration value="EQ"/>
              <xs:enumeration value="LTEQ"/>
              <xs:enumeration value="GTEQ"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Value" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Ownership">
    <xs:complexType>
      <xs:attribute name="status" type="xs:boolean" />
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Figure 7.12: The XML schema representation of our claims dialect.

for the policy. The ith such `All` clause in the policy should be processed as follows. First, a new rule will be created for this `All` clause:

```
(defrule rule-<i>
```

In the above rule, the `<i>` will be replaced with a counter indicating which `All` clause the rule represents. Assume this `All` clause has m `Token` elements. The kth such element will be processed as follows. First, a constraint will be added to the policy requiring that this token be presented:

```
  (credential (id ?id-<k>) (issuer ?iss-<k>) (owned ?o-<k>) (map ?m-<k>))
```

If this `Token`'s `Claims` element or `IssuerName` element specifies that the token must be issued by some specific issuer, `<issuer>`, the following test will be added to `rule-<i>`:

```
  (test (eq ?iss-<k> <issuer>))
```

If this `Token`'s `Claims` element contains the assertion `<Ownership Status="true">`, then the following test will be added to `rule-<i>`:

```
<wsp:Policy xmlns:wsp="..." xmlns:sp="...">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:X509Token xmlns:wsa="...">
        <sp:IssuerName>
          C=US/O=State University/OU=Registrar/CN=sts-reg.stateu.edu
        </sp:IssuerName>
        <wst:Claims Dialect="http://dais.cs.uiuc.edu/claim.xsd">
          <cl:Claim>
            <cl:Attribute>Type</cl:Attribute>
            <cl:Op>EQ</cl:Op>
            <cl:Value>Graduate Student</cl:Value>
          </cl:Claim>
          <cl:Ownership Status="true"/>
        </wst:Claims>
      </sp:X509Token>
      <sp:X509Token xmlns:wsa="...">
        <sp:IssuerName>
          C=US/O=ACM/CN=sts.acm.org
        </sp:IssuerName>
        <wst:Claims Dialect="http://dais.cs.uiuc.edu/claim.xsd">
          <cl:Claim>
            <cl:Attribute>MemberSince</cl:Attribute>
            <cl:Op>LTEQ</cl:Op>
            <cl:Value>2006</cl:Value>
          </cl:Claim>
          <cl:Ownership Status="true"/>
        </wst:Claims>
      </sp:X509Token>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Figure 7.13: A more complex example trust negotiation policy that makes use of our claim dialect.

```
(test (eq ?o-<k> true))
```

For all other constraint triples encoded in the `Claims` element of this token, the following test will be inserted. Here, `<op>` is either `eq`, `<`, `>`, `<=`, or `>=` depending on whether the operation encoded in the constraint tuple is `EQ`, `LT`, `GT`, `LTEQ`, or `GTEQ`. Similarly, `<name>` and `<value>` are placeholders for the attribute name and constraint value identified in the constraint triple.

```
(test (<op> (?m-<k> get <name>) <value>))
```

After each `Token` element in the ith `All` clause has been processed as above, `rule-<i>` will be terminated as follows:

```
=>
  (assert (satisfaction (resource-name rule-<i>)
          (credentials ?id-1 ... ?id-<m>))))
```

This process then repeats for each other `All` clause defined by the policy. Figure 7.14 shows a version of the policy from Figure 7.13 generated by using the above compilation rules. We now present the following theorem regarding the cor-

```
;; This policy is satisfied by graduate students at State University
;; who have been members of the ACM since at least 2006.
(defrule rule-1
  (credential (id ?id-1) (issuer ?iss-1) (owned ?o-1) (map ?m-1))
  (test (eq ?iss-1 "C=US/O=State University/OU=Registrar/CN=sts-reg.stateu.edu"))
  (test (eq ?o-1 true))
  (test (eq (?m-1 get "Type") "Graduate Student"))
  (credential (id ?id-2) (issuer ?iss-2) (owned ?o-2) (map ?m-2))
  (test (eq ?iss-2 "C=US/O=ACM/CN=sts.acm.org"))
  (test (eq ?o-2 true))
  (test (<= (?m-2 get "MemberSince") 2006))
=>
  (assert (satisfaction (resource-name rule-1)
          (credentials ?id-1 ?id-2))))
```

Figure 7.14: The policy presented in Figure 7.13 after being compiled for analysis by CLOUSEAU.

rectness and completeness of this compilation procedure:

THEOREM 7.7.1. *Assume that a trust negotiation policy p specified using the WS-Policy and WS-SecurityPolicy specifications is compiled using the above procedure into a* CLOUSEAU *policy p'. Given the policy p' and a set of security tokens S, the satisfying sets $s_1, \ldots, s_n$ returned by* CLOUSEAU *are* exactly *the subsets of S that satisfy the original policy p.*

Proof. Recall that the policy p is expressed in DNF form; that is, p is a disjunction of n conjunctive clauses. Each such disjunct identifies a unique set of security tokens that can be presented to satisfy p. For each disjunct d_i containing identifying m_i security tokens, the compilation process defined above creates one CLOUSEAU rule containing m_i patterns defined to match the tokens identified by d_i. Furthermore, for each security token t_j identified by d_i, the above process creates one `test` clause to check each claim constraint imposed on t_j by the policy p. Since no other rules are inserted into the policy p', CLOUSEAU cannot find any satisfying sets other than those identified by p when analyzing the policy p'. Similarly, since one such rule is created for each disjunct (i.e., satisfying set) in p, CLOUSEAU finds *all* satisfying sets in p when invoked on the policy p'. □

7.8 Summary

In this chapter, we described the design and implementation of the CLOUSEAU compliance checker. CLOUSEAU was designed to efficiently solve the type-3 compliance checker problem: given a set E of evidence and a policy p, determine all subsets of E that can be used to minimally satisfy p. Previous solutions for this

problem have had running time overheads that are exponential in the size of the union of all satisfying sets. CLOUSEAU's time overheads scale linearly in the size of the union of all satisfying sets for the tests conducted in [108], and as $O(NA)$ in general, where N is the number of satisfying sets for a policy and A is the average size of each satisfying set. On average, CLOUSEAU requires only tens of milliseconds to find all satisfying sets of credentials for a given policy.

CLOUSEAU achieves this level of performance by taking a non-traditional approach to theorem proving. Rather than directly analyzing access control policies written in high-level languages (such as Cassandra, RT, TPL, or $\mathcal{X}$-TNL), CLOUSEAU compiles high-level policies into an intermediate representation that specifies constraints on the actual credentials and other evidence that must be presented to gain access to a particular resource. CLOUSEAU then leverages Rete, an efficient pattern matching algorithm, to enumerate all satisfying sets. In this chapter, we showed that policies specified in the RT and WS-SecurityPolicy policy languages can be automatically compiled into the native rule format analyzed by CLOUSEAU, and proved that these compilation procedures were both complete and correct. Since all trust negotiation policies are eventually satisfied by these same types of evidence, we expect that equivalent compilation procedures could also be derived for other high-level policy languages. This allows policy writers to express their policies concisely using high-level policy languages, yet still analyze them efficiently.

Although CLOUSEAU is much more efficient than previous solutions to the type-3 compliance checker problem, further optimization is still an important area of future work. Spending tens of milliseconds during an interaction with their compliance checker is perfectly reasonable for entities in a peer-to-peer environment or clients in a client/server setting. However, servers that must process high volumes of traffic may need several such interactions for each trust negotiation session. An interesting direction of future work is to investigate high-level policy language constructs that can be compiled into CLOUSEAU policies that can be analyzed in a particularly efficient (or inefficient) manner. Better understanding of the language constructs that most directly affect compliance checker performance could help lead to the design of yet more efficient compliance checkers.

8 Related Work

This program was made possible, in part, by viewers like you.

—PBS

The work presented in this thesis is related to a wide spectrum of topics in computer science, including security, privacy, distributed computing, formal logic, and artificial intelligence. In this chapter, we highlight the bodies of work most directly related to the topics studied in this thesis. Section 8.1 identifies previous efforts by members of the security, privacy, and distributed computing communities related to the authorization system state consistency topics studied in Chapters 3 and 4. Topics related to the virtual fingerprinting, identity establishment, reputation, and audit problems studied in Chapter 5 are then discussed in Section 8.2. In Section 8.3, we overview research efforts contributing to the design and analysis of trust negotiation systems that directly influenced the TrustBuilder2 framework studied in Chapter 6. We conclude this chapter with Section 8.4, which discusses previous work that led to the design of the CLOUSEAU compliance checker presented in Chapter 7.

8.1 Safety, Consistency, and Concurrency Control

The concept of *safety* as it relates to trust negotiation has been discussed in several previous works, though the definitions of safety used in these works differ substantially from that considered in this thesis. Yu, Winslett, and Seamons [122] 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 [116] 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 [62] propose

another definition of safety based on the idea of information gain. The work presented in Chapters 3 and 4 of this thesis is orthogonal to these previous works. Specifically, our solutions are concerned with safety problems that emerge as a result of the inconsistency of the underlying state information used during policy evaluation, rather than those problems arising due to information leakage during an execution of any particular trust negotiation or distributed proof protocol. It would be prudent for system designers to consider both types of safety.

The Antigone Context Framework (ACF) provides a general-purpose framework for incorporating contextual data—such as room occupancy or a server's processing load—into authorization policy enforcement systems [87]. The ACF allows policy writers to incorporate contextual assertions into policies without requiring that the policy language include support for obtaining this data from the external world. Users of the ACF can write plug-ins for the framework that obtain this contextual information, which can then be accessed as policies are evaluated. These plug-ins could be used to enforce consistency constraints such as those discussed in Chapters 3 and 4, although it is unclear how one would enforce consistency constraints that depend on *all* contextual facts used in a given policy. By contrast, this thesis shows several ways in which the underlying authorization system can be used to enforce these types of constraints without requiring any involvement by policy writers.

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 [110], [31], 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 an incentive to cooperate. However, as was discussed in Sections 3.1 and 3.2.1, 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 thesis 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 pred-

icates over the system state is a well-known problem, to which an elegant solution was presented by Chandy and Lamport [32]. This algorithm is not directly applicable to the problem addressed in Chapters 3 and 4, however, due to the unstable nature of credential and fact statuses. There exist algorithms for collecting distributed state snapshots that can be used to evaluate unstable predicates (for a survey, see [4]), though these algorithms have very high overheads and make assumptions about process cooperation that are unreasonable given our highly-adversarial problem domain.

8.2 Audit and Reputation

Current research in reputation systems is orthogonally related to the problem solved by the Xiphos system in Chapter 5. While this area is too broad to survey in general, papers such as [51; 65; 85; 107] address the design of reputation systems for peer-to-peer and ad-hoc networks. These types of systems assume that entities have a single established identity in the system. Often times, these systems also suffer from whitewashing and ballot-stuffing attacks. To address false claims being inserted into the reputation system, the authors of [100] recommend designing reputation systems that require that non-repudiable *evidence* of a transaction be shown prior to inserting a reputation claim into the database. While this certainly prevents an entity from registering multiple claims, it requires that the underlying system (e.g., the grid computing system in our evaluation) support certified transactions. In Chapter 5, we presented a means of determining unique user identifiers in open systems where identity information is not always explicitly present and used these derived identifiers as a foundation for the Xiphos reputation system. To calculate the actual reputation values for entities in the system, we used equations similar to those defined in [85], though virtual fingerprints could be used in conjunction with *any* reputation calculation method. The nature of the virtual fingerprints derived using our method limits the damages that can be caused to Xiphos by the aforementioned attacks and does not require non-repudiable transaction support from the underlying system.

Other authors have also addressed the privacy versus trust trade-off that was discussed in Section 5.4. Anonymous credential schemes such as those presented in [29; 33; 34] assume that privacy is more important than any trust that can be established through history-based mechanisms (e.g., reputation systems). These systems provide a means for a credential issued to a given entity to be used under different pseudonyms to prevent transactions carried out by a single entity from ever being linked. In [106], the authors discuss this trade-off in detail and show

how entities can explicitly use multiple identities and allow linkages between these identities to be revealed to other parties to establish trust when needed. In Chapter 5, we allow system designers to balance this trade-off by choosing an appropriate deployment strategy for the Xiphos system. In addition, if Xiphos is used in systems supporting user-specified obligations, users can further limit the dissemination of their personal information, making the privacy versus trust trade-off more tunable.

A final area of work related to the discussion in Chapter 5 lies in the use of ontologies in trust management systems. In [80], the authors discuss how ontologies can be used to ease policy specification and administration in trust negotiation systems. In addition, they discuss how ontologies can be used to determine when certain types of information are being requested without need-to-know. In [85], the authors use service ontologies to add a context dimension to reputation ratings registered in their system. In our evaluation of Xiphos, we propose the use of credential ontologies while defining the $\gamma(\cdot)$ function used in our system. This use of ontologies is orthogonal to those presented in [80] and [85] and provides another way in which ontologies can simplify the management and strengthen the expressive power of trust management systems.

8.3 System Support for Trust Negotiation

As mentioned in Chapters 2 and 6, several research groups have developed implementations of trust negotiation systems. Winslett et al. developed the TrustBuilder system, which was the first such implementation [118]. TrustBuilder uses the IBM TE compliance checker for the TPL policy language [57] and supports the use of X.509 certificates [61]. Later, Bertino et al. implemented the Trust-$\mathcal{X}$ system [20], which expresses policies and credentials in the $\mathcal{X}$-TNL language [19]. Over time, this system evolved into the PP-Trust-$\mathcal{X}$ system described by Squicciarini et al., which supports additional privacy-preserving features [109]. Koshutanski and Massacci [67] describe a trust negotiation implementation designed for the web services environment that supports the use of X.509 certificates and SAML assertions [30] during the trust negotiation process. The common characteristic among these early trust negotiation implementations is that they all support a limited number of negotiation strategies, credential formats, and policy languages. This is in contrast to TrustBuilder2, which was designed to be a *framework* for trust negotiation that could be extended by its users to support any number of strategies, credential formats, policy languages, and other features.

The implementation most directly related to TrustBuilder2 is the system described by De Coi and Olmedilla [38]. As in this thesis, De Coi and Olmedilla came to the realization that a more unified architecture for trust negotiation systems is necessary to advance the trust negotiation state of the art. The authors examined the PEERTRUST [98] and PROTUNE [25] systems in an effort to derive a set of common requirements that should be supported by any trust negotiation implementation, and then implemented a framework embodying these requirements. While flexible, their system provides only a subset of the functionality embodied in TrustBuilder2. The reason for this is that the requirements for their system were derived by attempting to *unify* two existing systems, rather than by considering the trust negotiation problem space as a whole. As a result, the requirements derived by De Coi and Olmedilla are a subset of those derived in Sections 6.3 and 6.4 of this thesis. At present, we are involved in a collaboration with Olmedilla to develop a standard wire-level protocol for trust negotiation systems. Our hope is that by defining a standard communication protocol for these types of systems, we can further advance trust negotiation systems research by providing a means to develop interoperable trust negotiation implementations.

While developing implementations of trust negotiation systems is a necessary first step towards the eventual adoption of this approach to authorization, it is not the only barrier. Several research groups have investigated ways in which trust negotiation approaches to authorization can be integrated with existing protocols and distributed systems. Perhaps the most detailed study in this area was carried out by Hess et al. [58]. In this work, the authors investigated how the SSL/TLS protocol [45] could be extended to support trust negotiation natively. Such an extension would allow seamless integration of trust negotiation facilities with secure web browsing. This, in turn, would enable the use of increasingly expressive authorization and privacy policies on the World Wide Web. The authors show that only minor changes are needed to the SSL/TLS protocol standard to facilitate support for trust negotiation. They further provide a TLS implementation that supports trust negotiation natively by integrating the PureTLS [102] TLS implementation with the TrustBuilder trust negotiation system. Since this initial study, other groups have experimented with prototype trust negotiation systems in domains such as mobile computing [112], web services [39; 101], and grid computing [8; 40].

Unfortunately, directly modifying existing protocols to support trust negotiation is often a time-consuming and bureaucratic process, which will certainly hinder the adoption of this approach to authorization. To address this problem, Lee et al. describe Traust, a third-party authorization service that leverages the strengths of existing prototype trust negotiation systems [78; 79]. Traust acts as an authorization

broker that issues access tokens for resources in an open system after entities use trust negotiation to satisfy the appropriate resource access policies. The Traust architecture was designed to allow Traust to be integrated either directly with newer trust-aware applications or indirectly with existing legacy applications; this flexibility paves the way for the incremental adoption of trust negotiation technologies without requiring widespread software or protocol upgrades.

Lastly, we consider research efforts aimed at providing systems support for trust negotiation approaches to authorization. One problem that has been addressed in the research literature is that of credential management, as users often own multiple computing devices. Replicating credentials across multiple machines is problematic because it increases the probability that sensitive credentials will be stolen or leaked, and also requires users to keep several repositories up to date. In [8], Basney et al. propose to address this problem through the use of the MyProxy credential repository [7]. MyProxy was initially developed to manage the X.509 proxy certificates used in grid computing environments and provides a natural credential management system for the grid. As a further refinement of this approach, van der Horst and Seamons propose Thor: The Hybrid Online Repository [111]. Thor provides an intuitive user interface that allows users to securely manage their credentials by leveraging existing credential repositories, such as SACRED [56].

A final area in which researchers have investigated systems support for trust negotiation approaches to authorization is in responding to denial of service attacks. The existing research literature has focused on attacks resulting from the inclusion of spurious credential chains in the trust negotiation process [81; 104]. In Section 6.8.2 of this thesis, we showed that denial of service attacks can also be launched by taking advantage of the disparity between the cost of sending a policy to a remote party and the cost of actually analyzing the policy to determine whether it is satisfied. In [104], Ryutov et al. discuss a mechanism to make trust negotiation systems more adaptive in response to these types of attacks. They show that it is possible to integrate the TrustBuilder trust negotiation system with the Generic Authorization and Access-control API (GAA-API). The GAA-API can measure the system "context" through sensors such as intrusion detection systems (IDSes) and alter access control policies in response to changes in this context. Ryutov et al. argue that when an attack has been detected, changing the types of trust negotiation policies being used can help increase system availability. By leveraging more stringent policies during times of attack, questionable negotiations can be triaged earlier, thereby increasing the overall throughput of the authorization system.

8.4 Policy Compliance Checking

Over the years, the problem of checking compliance with security policies has become increasingly complex. Early access control, trust management, and network administration systems relied on type-1 compliance checkers that simply return a Boolean value indicating whether the policy in question is satisfied. Compliance checkers for the PolicyMaker [22; 23] and KeyNote [21] trust management systems are included in this first category, as the non-iterative nature of these systems makes the discovery of *why* a particular access was permitted superfluous; simply knowing that the compliance checker can construct a formal proof of authorization is sufficient. The CPOL compliance checker [26] is a highly-optimized compliance checker designed to enforce access policies on centralized resources in high-throughput environments, such as location-detection systems. CPOL uses aggressive caching and other optimizations to achieve incredible performance, but does not return evidence supporting the binary decisions that it makes. Lastly, the compliance checker for Ponder [42], which is used for policy-based network administration, also falls into this first category.

Recall that type-2 compliance checkers return one satisfying set of credentials in addition to a Boolean value, in the case that a policy is found to be satisfied. The compliance checker used by the REFEREE system [37] is capable of returning such justifications, though it need not do so. The ability to associate at least one satisfying set of credentials with a compliance checker decision is *required* by the trust negotiation process, as otherwise individuals could not determine which credentials should be sent to their negotiation partner after they determine that a remote policy can be satisfied. The compliance checkers for the XML-based policy languages $\mathcal{X}$-TNL [19] and the IBM Trust Policy Language [57] fall into this category, as do compliance checkers for existing logic-based trust negotiation policy languages, such as Cassandra [17] and the language presented by Koshutanski and Massacci in [68].

Type-3 compliance checkers extend the functionality provided by type-2 compliance checkers by returning *every* minimal set of credentials that can be used to satisfy a particular policy. Until now, no trust negotiation compliance checkers have been developed expressly for this purpose, although significant strategic benefits could be recognized by such a compliance checker. In [108], Smith et al. discuss several important uses of this type of compliance checker and describe the Satisfying Set Generation (SSgen) algorithm for discovering all satisfying sets for a given policy using a type-2 policy compliance checker. They show that when policies are expressed in disjunctive normal form (DNF), then a number of clever optimizations

can be made to prune the state space that must be searched for satisfying sets of credentials. They then evaluate the performance of an implementation of the SSgen algorithm that used the IBM TE compliance checker [57] as the base type-2 compliance checker. As shown in Chapter 7, CLOUSEAU presents a significant advance over this approach by formulating the compliance checking problem as a pattern matching problem. Efficient pattern matching algorithms can then be used to take a controlled forward chaining approach to policy compliance checking, which results in orders of magnitude better performance.

Recent work by Bauer et al. on the Grey system has also leveraged the advantages of forward chaining [13]. The Grey system is a prototype distributed proof system deployed at Carnegie Mellon University. One of the insights gained through the deployment and use of the Grey system is that the process of actually constructing a distributed proof can, at times, be very time consuming and cause delays that are unacceptable to the human user of the system. To increase the efficiency of the proof construction process, Bauer et al. have adopted a hybrid approach. Participants in the system use a forward chaining algorithm to determine all consequences of the facts in their local knowledge bases in an offline manner. At runtime, this pre-computed local knowledge is used in conjunction with a standard backward chaining prover to efficiently construct distributed proofs of authorization without requiring redundant local processing or the exploration of local "dead ends" in the proof process. CLOUSEAU differs from this work in three ways. First, the forward chaining done by CLOUSEAU is restricted in scope to the particular goal being proven. This provides an additional benefit over the approach in [13], since in our case, we are not interested in *all* facts that can be derived. Second, CLOUSEAU considers only proofs of authorization that can be constructed locally, while the theorem prover in [13] attempts to construct *distributed* proofs of authorization. Third, CLOUSEAU finds all ways that a particular policy can be satisfied, while Grey tries only to find a single proof of authorization. This is a result of the fact that credentials are not considered sensitive in the Grey system, so there is no advantage to discovering all possible proofs of authorization.

9 Conclusion

This is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.

—Winston Churchill

Recent years have seen the Internet become an increasingly open system in which users and resources from differing security domains have significant incentives to facilitate interactions with one another. The secure operation of such an open system requires an effective means of determining whether a given user is *authorized* to carry out the actions that he or she requests. Traditional approaches to the authorization problem rely on the use of identity-based access control lists to identify—by name—the users that are authorized to perform certain actions in the system. This approach works well in closed distributed computing environments that have well-known sets of users and resources. In open systems, however, resource owners do not necessarily know the users requesting access to their services and thus have no immediate way of inferring the attributes ascribed to a particular user when they learn his or her identity. As a result, identity-based approaches to authorization are ill-suited for use in open systems.

Researchers have addressed this shortcoming by developing decentralized and *attribute-based access control* (ABAC) approaches. Techniques such as trust negotiation and distributed proof construction allow user attributes and environmental context to be discovered at runtime and incorporated into the access control process on the fly. These approaches to authorization even allow users to protect their sensitive attributes from inadvertent disclosure and enable the establishment of *bilateral* trust between users and resources. To date, researchers have studied important aspects of decentralized ABAC approaches, including languages for expressing access control policies, strategies and tactics for constructing proofs of authorization, logics for reasoning about the outcomes of these types of protocols, and formal treatments of the information leakages that can occur during protocol execution. This research has developed a very strong theoretical foundation for distributed ABAC approaches to authorization, but has only scratched the surface of the multitude of systems-level problems that will accompany the eventual deployment of

these types of systems.

In this thesis, we showed that safely and securely adopting decentralized ABAC approaches to authorization requires careful consideration of the formal properties of these systems, as well as practical issues surrounding their deployment. To address the wide range of topics that must be considered, we answered a progression of important questions related to the safety analysis, deployment, implementation, and optimization of these types of systems. In the remainder of this chapter, we will first describe the contributions of this thesis in detail and show that our work has helped bridge the gap between the theory and practice of decentralized ABAC approaches to authorization. We then consider important avenues of future work.

9.1 Summary of Contributions

The majority of decentralized ABAC research to date has focused on building a strong theoretical foundation for these types of systems. As a result, the logical starting point for this thesis was to ask, *"Do existing theoretical models of trust negotiation and distributed proof protocols faithfully model asynchronous open systems, such as the Internet?"* A careful exploration of this question was the subject of Chapters Chapters 3 and 4, which led us to make the following contributions:

(1a) In Section 3.1, we presented a novel class of attacks in which an attacker can exploit the asynchronous nature of the Internet in combination with the autonomy granted to participants in decentralized ABAC protocols to subvert these protocols in ways not captured by the existing theory. In the scenarios that we describe, the attacker can steer the protocol execution so that the policy evaluator samples an inconsistent view of the system. This can result in an access control decision being made based upon incorrect user attribute or system context data. The net effect of this class of attacks is that accesses to system resources can—in certain circumstances—be permitted by decentralized ABAC protocols that would be denied by *any* centralized access control system.

(1b) In Section 3.2, we presented the first formalization of the view consistency problem for decentralized ABAC systems. By rigorously defining the notion of a system *view*, we were able to isolate the root causes of the above types of attacks, namely credential expiration, inter-credential dependencies, external events causing premature credential revocation, and unstable environments.

(1c) In Section 3.3, we defined four increasingly-stringent view consistency levels,

each of which is suited for different applications. The *incremental consistency* condition (Section 3.3.1) ensures that all credentials were valid when they were received by the policy evaluator. This level of consistency is sufficient for guaranteeing a transactional policy satisfaction semantics if the policy being satisfied consists exclusively of stable predicates. However, most policies contain unstable clauses (e.g., role memberships that may expire over time). The *internal consistency* condition (Section 3.3.2) addresses this by ensuring that there exists some time during the execution of the protocol at which all credentials were simultaneously valid. The *endpoint consistency* and *interval consistency* conditions (Section 3.3.3) are more stringent still, requiring that all credentials be valid simultaneously at the decision point of the protocol or throughout the entire protocol, respectively.

(1d) In Section 3.4, we developed lightweight distributed algorithms that can be used to enforce each of our view consistency levels in practice. We then proved the soundness of each of these algorithms when used in asynchronous and adversarial environments. In Sections 3.4 and 3.5, we further proved that these algorithms embody various privacy-preservation characteristics, have low computational and communication overheads, and can be used in conjunction with existing decentralized ABAC proposals.

(1e) Chapter 4 generalized the results from Chapter 3 to enforce view consistency conditions in decentralized ABAC systems in which the full details of a given authorization proof may not be available to the principal evaluating the satisfaction of the policy. An example of such a system is the distributed proof system developed by Minami and Kotz [90], in which portions of a proof tree can be hidden by confidentiality policies or inferences conducted over encrypted facts.

(1f) Sections 4.3.3 and 4.4.4 developed the notion of sliding windows of consistency. Given a user-defined window length Δ, this consistency condition allows the truth values of the facts making up a proof of authorization to fluctuate over time, so long as all facts are simultaneously true at least once in any given Δ time units. Sliding windows of consistency are useful for enforcing access control conditions over long periods of time in systems where some degree of fluctuation or uncertainty is expected. This is appealing in pervasive computing environments in which the context of the system may fluctuate often around some acceptable baseline.

(1g) As in Chapter 3, we proved the soundness of each consistency enforcement algorithm developed in Chapter 4. Furthermore, each algorithm was imple-

mented as an extension to the Minami-Kotz proof system and shown to have minimal runtime overheads (Section 4.5) and proven to maintain the semantics of the original proof system (Section 4.4).

The above contributions demonstrate that although the existing decentralized ABAC theory does not account for all of the nuances of asynchronous distributed systems, they can be accounted for in practice. We showed that existing decentralized ABAC protocols can be *wrapped* with consistency enforcement code that accounts for this gap between theory and practice without imposing heavy computation or communication overheads.

After showing that existing decentralized ABAC protocols can be *safely* deployed in asynchronous systems, we considered the impacts of deploying these types of authorization systems on existing audit practices. Specifically, we asked *"How can users be held accountable for their actions in a system without concrete user identities?"* Chapter 5 focused on developing a scalable means of tracking and auditing users in decentralized ABAC systems. To this end, we made the following contributions:

(2a) In Section 5.2, we showed that the set of credentials disclosed by an individual during a decentralized ABAC protocol forms one unique *description* of that individual. As a result, two overlapping descriptions necessarily refer to the same individual and thus can be used to audit the actions taken by this individual over time. Since a user's credentials are often considered sensitive, however, descriptions should not be used to audit actions taken by a user across multiple domains, as this could lead to privacy violations. To overcome this barrier, we developed the notion of *virtual fingerprints*, which are opaque identifiers derived from user descriptions that can be used in interdomain audit scenarios without leaking a user's sensitive attribute information.

(2b) To demonstrate the utility of services based on virtual fingerprints, we developed Xiphos, a reputation system that is keyed on virtual fingerprints instead of traditional user identities. In Section 5.3, we described how Xiphos can be deployed as a centralized, totally decentralized, or hierarchical (super-peer) reputation service. In Section 5.4.1, we explored the privacy and performance trade-offs that these deployments allow system designers to balance.

(2c) In Section 5.4.2, we showed that the use of Xiphos makes certain attacks against reputation systems difficult to carry out for habitual cheaters. Specifically, the whitewashing, slander, and self-promotion classes of attacks are prevented due to the difficulty of establishing new virtual fingerprints. Furthermore, the Sybil attack is mitigated to a great extent, as users cannot create an unbounded number of identities within the Xiphos system.

(2d) Section 5.5 evaluated the scalability of audit services based on virtual fingerprints by examining the performance of Xiphos in simulated grid computing systems ranging in size from 10,000–70,000 nodes. We showed that query throughput remained high and local reputation database sizes remained manageable even in the largest of these systems. This demonstrates that despite having a highly expressive identity matching semantics, systems based on virtual fingerprints remain scalable in non-trivial usage scenarios.

Our study of virtual fingerprints shows that the actions of users in decentralized ABAC systems can be audited over time, even in systems spanning multiple organizational domains. When coupled with our previous results describing a mechanism for facilitating the deployment of trust negotiation systems in legacy environments [78; 79], this indicates that decentralized ABAC systems are well-suited for test deployments in existing distributed systems.

At this stage of the thesis, our focus shifted to more practical issues surrounding the implementation and optimization of trust negotiation systems. To begin our study of this area, we asked, *"What are the key systems and architectural characteristics of the trust negotiation process, and what are their impacts on the performance of an organization's authorization infrastructure?"* In Chapter 6, we studied this question in detail and made the following contributions:

(3a) In Section 6.3, we carried out a detailed use case analysis to study several potential deployment scenarios for decentralized ABAC systems. In Section 6.4, we used the characteristics of these systems to derive a set of functional requirements for trust negotiation systems. We showed that existing trust negotiation systems provide support for less than one third of the requirements that we identified.

(3b) In Section 6.5, we describe the TrustBuilder2 framework for trust negotiation, which was designed to meet the functional requirements uncovered during our use case analysis. TrustBuilder2 makes use of an extensible data type hierarchy and a flexible communication protocol to become the first fully-configurable framework for the design, deployment, and analysis of trust negotiation protocols. The design of TrustBuilder2 demonstrates that a large number of trust negotiation proposals in the research literature can be unified under a single system architecture.

(3c) TrustBuilder2 demonstrates that adding a high degree of flexibility to advanced authorization frameworks does not necessarily lead to high runtime overheads. In Section 6.7, we showed that the time spent handling the indirection needed to support user plug-ins and other extensions amounts to less

than 0.2% of the total execution time.

(3d) Studies conducted using TrustBuilder2 led us to identify the primary bottlenecks in the trust negotiation process (Section 6.7) and uncover a novel class of denial of service attacks against trust negotiation systems in which the cost of policy analysis is exploited (Section 6.8.2).

The analysis that we conducted using TrustBuilder2 showed that checking compliance with ABAC policies is the most expensive portion of the trust negotiation process. This led us to pose the question, *"Is it possible to increase the efficiency of the compliance checking process without altering its completeness or correctness properties?"* We explore this question in Chapter 7 and make the following contributions during our study:

(4a) We showed that the inefficiencies of the compliance checking process stem from the fact that, for historical reasons, the policy compliance checking problem is often formulated as a theorem proving problem. When analyzing decentralized ABAC policies, it is beneficial to find *all* proofs of authorization so that a locally-optimal choice can be made to advance the state of the authorization protocol. While a theorem-proving approach to policy compliance checking is very natural, it carries with it undesirable overheads, as theorem provers are generally used to find a single proof of each target theorem. As a result, compliance checkers based upon theorem provers tend to perform poorly for generating multiple proofs of authorization.

(4b) In Section 7.4, we developed the CLOUSEAU compliance checker, which uses a pattern matching formalism to efficiently find *all* ways in which a policy can be satisfied. The evaluation conducted in Section 7.5 showed that this approach outperforms existing theorem-proving approaches to compliance checking by orders of magnitude. As a concrete point of comparison, the compliance checker analyzed in [108] takes over 10 seconds to find two overlapping satisfying sets containing a total of 20 credentials, while CLOUSEAU finds the same satisfying sets in approximately 40 ms.

(4c) CLOUSEAU improves not only the efficiency of the trust negotiation process, but also its utility. By determining all satisfying sets for a given policy, CLOUSEAU uncovers the entire "next-step" state space for any given trust negotiation. This allows negotiation strategies to make more intelligent decisions regarding how to proceed.

(4d) In Sections 7.6 and 7.7, we demonstrate that existing policy languages can be compiled into a format suitable for analysis by CLOUSEAU. Section 7.6

describes a compilation process for the RT_0 policy language and provides a formal proof of correctness and completeness for CLOUSEAU when operating on compiled RT_0 policies. In Section 7.6.4, we show that this compilation procedure can be extended in a very natural manner to support RT_1 policies, which allow for the definition of parameterized roles. Section 7.7 describes a compilation process for policies written in the WS-SecurityPolicy language, and again provides a proof of correctness and completeness. These compilation procedures imply that policy writers can take advantage of the existing policy languages that best meet their needs without negatively influencing the runtime characteristics of trust negotiation implementations.

Recall from Chapter 1 that our goal in this thesis was to provide adequate evidence to justify the following thesis statement:

It is possible to develop extensible and efficient decentralized attribute-based access control systems that can be safely deployed in large-scale asynchronous distributed environments.

Contributions (1a)–(1g) provide proof that decentralized ABAC approaches to authorization can *safely* be used in asynchronous distributed environments. Contributions (2a)–(2d) show that despite a relaxed notion of user identity, the actions taken by users in systems relying on decentralized ABAC systems can, in fact, be audited without leaking sensitive attribute information. When coupled with the results in [78; 79], this implies that decentralized ABAC systems can be *deployed* without breaking compatibility with existing services and best practices. Contributions (3a)–(3d) show that *flexible* and *extensible* decentralized ABAC systems can be developed without imposing high overheads. Finally, contributions (4a)–(4d) show that these systems can be greatly optimized to provide an *efficient* and *expressive* means of access control for open systems.

9.2 Future Work

This thesis focused on bridging the gap between the theory and practice of decentralized ABAC systems. The long term success of these systems depends on a number of other issues, however. First and foremost, test deployments of these types of systems are necessary to uncover future research challenges relating to the adoption of these systems. At Carnegie Mellon University, researchers have deployed the Grey distributed proof system [11] and have identified a number of

important topics related to the efficiency, usability, and management of distributed proof systems. To date, no such study has been conducted involving the use of trust negotiation protocols. Our hope, however, is that the TrustBuilder2 framework that was developed in this thesis will provide the necessary system support for such a study to take place. Along the same lines, it is important to investigate the deployment of *multiple* approaches to trust negotiation and examine the interoperability issues that will inevitably arise. As mentioned in Section 8.3, we are currently involved in an effort to develop a standardized wire-level protocol for trust negotiation systems to facilitate the development of such interoperable trust negotiation implementations.

With respect to the deployment of decentralized ABAC systems, it is also important to examine the scalability of these systems under heavy load. As was discussed in Chapter 6, the trust negotiation process can take multiple rounds of communication and hundreds of milliseconds to complete; this is much more expensive than simply checking the correctness of a username/password pair. If decentralized ABAC approaches are to be successfully deployed in open systems with large numbers of users, approaches to improve the throughput of these systems must be examined. One possible approach to solving this problem involves delegating responsibility for carrying out the interactive portions of these protocols to multiple *helper nodes*. After a successful protocol execution, a helper node could then generate a receipt for the interaction that includes the target policy that was satisfied, as well as the collection of credentials used to satisfy the policy. This receipt can then be passed to the reference monitor for a particular resource, which will permit access if and only if the correctness and validity of the receipt can be verified. As was shown in Chapter 7, an optimized compliance checker can check the correctness of such a receipt in tens of milliseconds or faster. The use of multiple helper nodes in conjunction with a load balancing system could thus greatly increase the throughput of decentralized ABAC systems, provided that a safe and restricted means of delegating the usage of credentials to helper nodes can be developed.

Perhaps the most important barrier to the eventual success of decentralized ABAC systems is their usability. Studies have shown that users have a difficult time using even a single digital certificate correctly [114]. However, in trust negotiation and distributed proof approaches to authorization, users must manage a large collection of certificates attesting to their various attributes and capabilities. The credential management systems discussed in Section 8.3 are a good first step towards addressing this problem, but further user studies are necessary. A related problem that must be addressed is that of policy specification and management. Very few computer users are likely to possess the mathematical sophistication to correctly

author and interpret Datalog-based security policies. Developing intuitive interfaces for specifying, editing, and deploying policies will be an interesting challenge likely to require experts from both the security and human-computer interaction fields. A particularly challenging aspect of this problem will lie in quantifying and explaining the differences between versions of a policy as it is being modified by the user. Ensuring that seemingly-benign policy modifications do not have unintended consequences will surely be a challenging area of future research.

Another important area to investigate is the development of more flexible models of access control and authorization. In trust negotiation and distributed proof systems, access to some resource R is granted if and only if the policy protecting R is satisfied exactly. While such requirements provide very strong guarantees regarding the conditions under which accesses are granted, this requires that the policies protecting any given resource consider *every* possible situation under which access should be granted. A recent report by the JASON Program Office shows that within the intelligence domain, access control policies often do not (and cannot) consider all such cases [63]. As a result, people fail to use existing access control and document classification systems properly, as doing so would hinder their ability to carry out their jobs. In response to this realization, the notion of risk-based access control has been proposed to help prevent systems from blocking authorized users during unexpected circumstances [35; 63]. Risk-based access control addresses this problem by quantifying the risk associated with access to a particular piece of data. Organizations then receive risk budgets that they can allocate to their employees to allow them to purchase information that they would not normally be allowed to access, using risk tokens.

While this approach is interesting, there are two major difficulties that must be overcome. First, it is not obvious how to price access to information resources. Second, these systems do not have the notion of a formal proof of authorization that is present in distributed proof and trust negotiation systems. To address these two problems, we are just beginning to study ways in which risk-based access control approaches can be unified with decentralized ABAC approaches to authorization. This approach will combine the formal guarantees of decentralized ABAC approaches with the ability to utilize risk-based economic models that enable the "fuzzing" of access control policies to account for unpredicted, yet authorized, flows of information. Such an approach would allow principals to build *approximate*—rather than strict—proofs of authorization when they attempt to access a given resource. Dynamic pricing models would then be used to quantify the difference between the expected proof of authorization (as defined by the access control policy) and the proof generated by the requesting principal. If the cost of access is

determined to be below a certain threshold, the requesting principal would then have the opportunity to purchase access to the resource using risk tokens. Such an access control model still produces formal proofs of authorization (which will be useful for a posteriori audits) while also allowing for some flexibility in unexpected situations. We further hope to show that pricing the *difference* between proofs of access will be an easier task than pricing access to data in general, as the pricing strategy can incorporate the approximate proof that was generated.

References

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

[2] Jalal Al-Muhtadi, Anand Ranganathan, Roy Campbell, and Dennis Mickunas. Cerberus: a context-aware security scheme for smart spaces. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, pages 489–496, March 2003.

[3] Andrew W. Appel and Edward W. Felten. Proof-carrying authentication. In *Proceedings of the Sixth ACM Conference on Computer and Communications Security (CCS)*, pages 52–62, November 1999.

[4] Ozalp Babaoğlu and Keith Marzullo. Consistent global states of distributed systems: Fundamental concepts and mechanisms. In Sape 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.

[5] Jean Bacon, Ken Moody, and Walt Yao. A model of OASIS role-based access control and its support for active security. *ACM Transactions on Information and System Security*, 5(4):492–540, 2002.

[6] David Balenson. Privacy enhancement for internet electronic mail: Part III: Algorithms, modes, and identifiers. IETF RFC 1423, February 1993.

[7] Jim Basney, Marty Humphrey, and Von Welch. The MyProxy online credential repository. *Software: Practice and Experience*, 35(9):801–816, July 2005.

[8] Jim Basney, Wolfgang Nejdl, Daniel Olmedilla, Von Welch, and Marianne Winslett. Negotiating trust on the Grid. In *Proceedings of the Second WWW Workshop on Semantics in P2P and Grid Computing*, May 2004.

[9] Lujo Bauer, Lorrie Cranor, Robert W. Reeder, Michael K. Reiter, and Kami Vaniea. A user study of policy creation in a flexible access-control system. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, April 2008.

[10] Lujo Bauer, Lorrie Faith Cranor, Michael K. Reiter, and Kami Vaniea. Lessons learned from the deployment of a smartphone-based access-control system. In *Proceedings of the 3rd Symposium on Usable Privacy and Security (SOUPS)*, pages 64–75, July 2007.

[11] Lujo Bauer, Scott Garriss, Jonathan M. McCune, Michael K. Reiter, Jason Rouse, and Peter Rutenbar. Device-enabled authorization in the Grey system. In *Proceedings of Information Security, Eighth International Conference (ISC 2005)*, volume 3650 of *Lecture Notes in Computer Science*, pages 431–445. Springer-Verlag, September 2005.

[12] Lujo Bauer, Scott Garriss, and Michael K. Reiter. Distributed proving in access-control systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 81–95, May 2005.

[13] Lujo Bauer, Scott Garriss, and Michael K. Reiter. Efficient proving for practical distributed access-control systems. In *Proceedings of the 12th European Symposium on Research in Computer Security (ESORICS 2007)*, volume 4734 of *Lecture Notes in Computer Science*, pages 19–37. Springer, September 2007.

[14] Lujo Bauer, Scott Garriss, and Michael K. Reiter. Detecting and resolving policy misconfigurations in access-control systems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT)*, June 2008.

[15] Lujo Bauer, Michael A. Schneider, and Edward W. Felten. A general and flexible access-control system for the Web. In *Proceedings of the 11th USENIX Security Symposium*, pages 93–108, August 2002.

[16] Moritz Y. Becker. A formal security policy for an NHS electronic health record service. Technical Report UCAM-CL-TR-628, University of Cambridge Computer Laboratory, March 2005.

[17] Moritz Y. Becker and Peter Sewell. Cassandra: Distributed access control policies with tunable expressiveness. In *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '04)*, pages 159–168, 2004.

[18] Moritz Y. Becker and Peter Sewell. Cassandra: flexible trust management, applied to electronic health records. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, pages 139–154, 2004.

[19] Elisa Bertino, Elena Ferrari, and Anna Cinzia Squicciarini. $\mathcal{X}$ -TNL: An XML-based language for trust negotiations. In *Proceedings of the Fourth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '03)*, pages 81–84, June 2003.

[20] Elisa Bertino, Elena Ferrari, and Anna Cinzia Squicciarini. Trust-$\mathcal{X}$: A peer-to-peer framework for trust establishment. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):827–842, July 2004.

[21] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. KeyNote: Trust management for public-key infrastructures (position paper). *Lecture Notes in Computer Science*, 1550:59–63, 1999.

[22] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the IEEE Conference on Security and Privacy*, pages 164–173, May 1996.

[23] Matt Blaze, Joan Feigenbaum, and Martin Strauss. Compliance checking in the PolicyMaker trust management system. In *Proceedings of the Second International Conference on Financial Cryptography*, number 1465 in Lecture Notes in Computer Science, pages 254–274. Springer, February 1998.

[24] Piero Bonatti and Pierangela Samarati. Regulating service access and information release on the Web. In *Proceedings of the Seventh ACM Conference on Computer and Communications Security*, pages 134–143, November 2000.

[25] Piero A. Bonatti and Daniel Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *Proceedings of the Sixth IEEE Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, pages 14–23, June 2005.

[26] Kevin Borders, Xin Zhao, and Atul Prakash. CPOL: High-performance policy evaluation. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS)*, pages 147–157, November 2005.

[27] Nikita Borisov and Kazuhiro Minami. Single-bit re-encryption with applications to distributed proof systems. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 48–55, November 2007.

[28] Kevin D. Bowers, Lujo Bauer, Deepak Garg, Frank Pfenning, and Michael K. Reiter. Consumable credentials in logic-based access-control systems. In *Proceedings of the 2007 Network and Distributed System Security Symposium (NDSS)*, pages 143–157, February 2007.

[29] Jan Camenisch and Els Van Herreweghen. Design and implementation of the *idemix* anonymous credential system. In *Proceedings of the Ninth ACM Conference on Computer and Communications Security*, pages 21–30, November 2002.

[30] Scott Cantor, John Kemp, Rob Philpott, and Eve Maler (Editors). Assertions and protocols for the OASIS security assertion markup language (SAML V2.0). OASIS Standard, March 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[31] Wojciech Cellary, Erol Gelenbe, and Tadeusz Morzy. *Concurrency Control in Distributed Database Systems*. Elsevier Science Publishing Company, Inc., 1988.

[32] K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985.

[33] David Chaum. Security without identification: Transaction systems to make Big Brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[34] David Chaum and Jan-Hendrik Evertse. A secure and privacy-protecting protocol for transmitting personal information between organizations. In *CRYPTO '88*, volume 263 of *LNCS*, pages 118–167. Springer-Verlag, 1988.

[35] Pau-Chen Cheng, Pankaj Rohatgi, Claudia Keser, Paul A. Karger, Grant M. Wagner, and Angela Schuett Reninger. Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 222–230, May 2007.

[36] David R. Cheriton and Dale Skeen. Understanding the limitations of causally and totally ordered communication. In *Proceedings of the 14th ACM Symposium on Operating Systems Priniciples*, pages 44–57, December 1993.

[37] Yang-Hua Chu, Joan Feigenbaum, Brian LaMacchia, Paul Resnick, and Martin Strauss. REFEREE: Trust management for web applications. *World Wide Web Journal*, 2(3):127–139, Summer 1997.

[38] Juri L. De Coi and Daniel Olmedilla. A flexible policy-driven trust negotiation model. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 450–453, November 2007.

[39] Juri L. De Coi, Daniel Olmedilla, Sergej Zerr, Piero A. Bonatti, and Luigi Sauro. A trust management package for policy-driven protection & personalization of Web content. In *Proceedings of the IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2008)*, June 2008.

[40] Ionut Constandache, Daniel Olmedilla, and Frank Siebenlist. Policy-driven negotiation for authorization in the Grid. In *Proceedings of the IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2007)*, pages 211–220, June 2007.

[41] Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind K. Dey, Mustaque Ahamad, and Gregory D. Abowd. Securing context-aware applications using environment roles. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, pages 10–20, May 2001.

[42] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The Ponder policy specification language. In *Proceedings of the Second IEEE Workshop on Policies for Distributed Systems and Networks (POLICY)*, number 1995 in Lecture Notes in Computer Science, pages 18–39. Springer, January 2001.

[43] Data Encryption Standard (DES). Federal Information Processing Standard FIPS PUB 46-3, October 1999.

[44] John DeTreville. Binder, a logic-based security language. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 105–113, May 2002.

[45] Tim Dierks and Christopher Allen. The TLS protocol version 1.0. IETF Request for Comments RFC 2246, January 1999.

[46] Eclipse—an open development platform. Web site, January 2007. http://www.eclipse.org.

[47] Eclipse test and performance tools platform. Web site, January 2007. http://www.eclipse.org/tptp.

[48] Jeffrey Schlimmer (Editor). Web Services Policy 1.2—Framework (WS-Policy). W3C Member Submission, April 2006. http://www.w3.org/Submission/WS-Policy/.

[49] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI certificate theory. IETF Request for Comments RFC 2693, September 1999.

[50] Chris Faloutsos. Access methods for text. *ACM Computing Surveys*, 17(1):49–74, 1985.

[51] Alberto Fernandes, Evangelos Kotsovinos, Sven Östring, and Boris Dragovic. Pinocchio: Incentives for honest participation in distributed trust management. In *Proceedings of the Second International Conference on Trust Management (iTrust 2004)*, pages 63–77, March 2004.

[52] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 27(3):219–227, 1985.

[53] Ernest Friedman-Hill. Jess: The rule engine for the Java platform. Web site, April 2007. ⟨http://www.jessrules.com⟩.

[54] Deepak Garg, Lujo Bauer, Kevin D. Bowers, Frank Pfenning, and Michael K. Reiter. A linear logic of authorization and knowledge. In *Proceedings of the 11th European Symposium on Research in Computer Security (ESORICS)*, volume 4189 of *Lecture Notes in Computer Science*, pages 297–312. Springer, September 2006.

[55] Carl A. Gunter and Trevor Jim. Policy-directed certificate retrieval. *Software—Practice and Experience*, 30(15):1609–1640, 2000.

[56] Dale Gustafson, Mike Just, and Magnus Nystrom. Securely available credentials (SACRED)—credential server framework. IETF Request for Comments RFC 4120, April 2004.

[57] Amir Herzberg, Yosi Mass, Joris Michaeli, Dalit Naor, and Yiftach Ravid. Access control meets public key infrastructure, or: assigning roles to strangers. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 2–14, May 2000.

[58] Adam Hess, Jared Jacobson, Hyrum Mills, Ryan Wamsley, Kent E. Seamons, and Bryan Smith. Advanced client/server authentication in TLS. In *Proceedings of the Network and Distributed Systems Security Symposium*, February 2002.

[59] Kevin Hoffman, David Zage, and Cristina Nita-Rotaru. A survey of attack and defense techniques for reputation systems. *ACM Computing Surveys*, to appear.

[60] Jason Holt, Robert W. Bradshaw, Kent E. Seamons, and Hilarie K. Orman. Hidden credentials. In *Proceedings of the Second ACM Workshop on Privacy in the Electronic Society*, pages 1–8, October 2003.

[61] Russel Housely, Warwick Ford, Tim Polk, and David Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. IETF Request for Comments RFC-2459, January 1999.

[62] Keith Irwin and Ting Yu. Preventing attribute information leakage in automated trust negotiation. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, pages 36–45, November 2005.

[63] Horizontal integration: Broader access models for realizing information dominance. Technical Report JSR-04-132, JASON Program Office, MITRE Corporation, December 2004.

[64] Trevor Jim. SD3: A trust management system with certified evaluation. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 106–115, May 2001.

[65] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th International Conference on World Wide Web (WWW'03)*, pages 640–651, May 2003.

[66] Vladimir Kolovski, James Hendler, and Bijan Parsia. Analyzing web access control policies. In *Proceedings of the 16th International World Wide Web Conference (WWW)*, pages 677–686, May 2007.

[67] Hristo Koshutanski and Fabio Massacci. Interactive access control for web services. In *Proceedings of the 19th IFIP Information Security Conference (SEC)*, pages 151–166, August 2004.

[68] Hristo Koshutanski and Fabio Massacci. An interactive trust management and negotiation scheme. In *Proceedings of the Second International Workshop on Formal Aspects in Security and Trust (FAST)*, pages 139–152, August 2004.

[69] Hristo Koshutanski and Fabio Massacci. Interactive credential negotiation for stateful business processes. In *Proceedings of the Third International Conference on Trust Management (iTrust)*, pages 257–273, May 2005.

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

[71] Adam J. Lee, Kazuhiro Minami, and Marianne Winslett. Lightweight consistency enforcement schemes for distributed proofs with hidden subtrees. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT 2007)*, pages 101–110, June 2007.

[72] Adam J. Lee, Kazuhiro Minami, and Marianne Winslett. On the consistency of distributed proofs with hidden subtrees. *ACM Transactions on Information and System Security (TISSEC)*, submitted.

[73] Adam J. Lee and Marianne Winslett. Safety and consistency in policy-based authorization systems. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS 2006)*, pages 124–133, November 2006.

[74] Adam J. Lee and Marianne Winslett. Virtual fingerprinting as a foundation for reputation in open systems. In *Proceedings of the Fourth International Conference on Trust Management (iTrust 2006)*, pages 236–251, May 2006.

[75] Adam J. Lee and Marianne Winslett. Towards an efficient and language-agnostic compliance checker for trust negotiation systems. In *Proceedings of the Third ACM Symposium on Information, Computer and Communications Security (ASIACCS 2008)*, pages 228–239, March 2008.

[76] Adam J. Lee and Marianne Winslett. Towards standards-compliant trust negotiation for web services. In *Proceedings of the Joint iTrust and PST Conferences on Privacy, Trust Management, and Security (IFIPTM 2008)*, June 2008.

[77] Adam J. Lee and Marianne Winslett. Enforcing safety and consistency constraints in policy-based authorization systems. *ACM Transactions on Information and System Security (TISSEC)*, to appear.

[78] Adam J. Lee, Marianne Winslett, Jim Basney, and Von Welch. Traust: A trust negotiation-based authorization service for open systems. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT 2006)*, pages 39–48, June 2006.

[79] Adam J. Lee, Marianne Winslett, Jim Basney, and Von Welch. The Traust authorization service. *ACM Transactions on Information and System Security (TISSEC)*, 11(1), February 2008.

[80] Travis Leithead, Wolfgang Nejdl, Daniel Olmedilla, Kent E. Seamons, Marianne Winslett, Ting Yu, and Charles C. Zhang. How to exploit ontologies in trust negotiation. In *Proceedings of the Third International Semantic Web Conference Workshop on Trust, Security, and Reputation on the Semantic Web*, November 2004.

[81] Jiangtao Li, Ninghui Li, Xiaofeng Wang, and Ting Yu. Denial of service attacks and defenses in decentralized trust management. In *Proceedings of the Second International Conference on Security and Privacy in Communication Networks (SecureComm)*, August 2006.

[82] Jiangtao Li, Ninghui Li, and William 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, November 2005.

[83] Ninghui Li and John Mitchell. RT: A role-based trust-management framework. In *Proceedings of the Third DARPA Information Survivability Conference and Exposition*, April 2003.

[84] Ninghui Li, William H. Winsborough, and John C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, 2003.

[85] Jinshan Liu and Valérie Issarny. Enhanced reputation mechanism for mobile ad hoc networks. In *Proceedings of the Second International Conference on Trust Management (iTrust 2004)*, pages 48–62, March 2004.

[86] Stephen Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, University of Stirling, 1994.

[87] Patrick McDaniel. On context in authorization policy. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*, pages 80–89, June 2003.

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

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

[90] Kazuhiro Minami and David Kotz. Secure context-sensitive authorization. *Journal of Pervasive and Mobile Computing (PMC)*, 1(1):123–156, March 2005.

[91] Kazuhiro Minami and David Kotz. Scalability in a secure distributed proof system. In *Proceedings of the Fourth International Conference on Pervasive Computing (Pervasive)*, pages 220–237, May 2006.

[92] Daniel P. Miranker. TREAT: A better match algorithm for AI production systems. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 42–47, August 1987.

[93] Tim Moses. XACML 2.0 Core: eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard, February 2005. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[94] Michael Myers, Rich Ankney, Ambarish Malpani, Slava Glaperin, and Carlisle Adams. X.509 Internet public key infrastructure online certificate status protocol - OCSP. IETF RFC 2560, June 1999.

[95] Ginger Myles, Adrian Friday, and Nigel Davies. Preserving privacy in environments with location-based applications. *IEEE Pervasive Computing*, 2(1):56–64, January–March 2003.

[96] Anthony Nadalin, Marc Goodner, Martin Gudgin, Abbie Barbir, and Hans Granqvist (Editors). WS-SecurityPolicy 1.2. OASIS Standard, July 2007. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/.

[97] Anthony Nadalin, Marc Goodner, Martin Gudgin, Abbie Barbir, and Hans Granqvist (Editors). WS-Trust 1.3. OASIS Standard, March 2007. http://docs.oasis-open.org/ws-sx/ws-trust/200512/.

[98] Wolfgang Nejdl, Daniel Olmedilla, and Marianne Winslett. Peertrust: Automated trust negotiation for peers on the semantic web. In *Proceedings of the VLDB Workshop on Secure Data Management (SDM)*, volume 3178 of *Lecture Notes in Computer Science*, pages 118–132. Springer, August 2004.

[99] Clifford Neuman, Tom Yu, Sam Hartman, and Kenneth Raeburn. The Kerberos network authentication service (V5). IETF Request for Comments RFC 4120, July 2005.

[100] Philipp Obreiter. A case for evidence-aware distributed reputation systems. In *Proceedings of the Second International Conference on Trust Management (iTrust 2004)*, pages 33–47, March 2004.

[101] Lars E. Olson, Marianne Winslett, Gianluca Tonti, Nathan Seeley, Andrzej Uszok, and Jeffrey Bradshaw. TrustBuilder as an authorization service for web services. In *Proceedings of the International Workshop on Security and Trust in Decentralized/Distributed Data Structures (STD3S)*, April 2006.

[102] Eric Rescorla. Claymore PureTLS. Web site, July 2005. http://www.rtfm.com/puretls/.

[103] Ronald L. Rivest. The MD5 message-digest algorithm. IETF RFC 1321, April 1992.

[104] Tatyana Ryutov, Li Zhou, Clifford Neuman, Travis Leithead, and Kent E. Seamons. Adaptive trust negotiation and access control. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*, pages 139–146, June 2005.

[105] Kent E. Seamons, Marianne Winslett, Ting Yu, Bryan Smith, Evan Child, Jared Jacobson, Hyrum Mills, and Lina Yu. Requirements for policy languages for trust negotiation. In *Proceedings of the Third IEEE Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 68–79, June 2002.

[106] Jean-Marc Seigneur and Christian Damsgaard Jensen. Trading privacy for trust. In *Proceedings of the Second International Conference on Trust Management (iTrust 2004)*, pages 93–107, March 2004.

[107] Ali Aydin Selcuk, Ersin Uzun, and Mark Resat Pariente. A reputation-based trust management system for P2P networks. In *Proceedings of the Fourth IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2004)*, pages 251–258, April 2004.

[108] Bryan Smith, Kent E. Seamons, and Michael D. Jones. Responding to policies at runtime in TrustBuilder. In *Proceedings of the Fifth IEEE Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 149–158, June 2004.

[109] Anna Squicciarini, Elisa Bertino, Elena Ferrari, Federica Paci, and Bhavani Thuraisingham. PP-Trust-$\mathcal{X}$: A system for privacy preserving trust negotiations. *ACM Transactions on Information and System Security*, 10(3), July 2007.

[110] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, Upper Saddle River, New Jersey, 2002.

[111] Timothy W. van der Horst and Kent E. Seamons. Short paper: Thor—the hybrid online repository. In *Proceedings of the First IEEE International Conference on Security and Privacy for Emerging Areas in Communications Networks*, September 2005.

[112] Timothy W. van der Horst, Tore Sundelin, Kent E. Seamons, and Charles D. Knutson. Mobile trust negotiation: Authentication and authorization in dynamic mobile networks. In *Proceedings of the Eighth IFIP Conference on Communications and Multimedia Security*, September 2004.

[113] Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. A logic-based framework for attribute based access control. In *Proceedings of the Second ACM Workshop on Formal Methods in Security Engineering (FMSE 2004)*, pages 45–55, October 2004.

[114] Alma Whitten and J.D. Tygar. Why Johnny can't encrypt: A usability case study of PGP 5.0. In *Proceedings of the Eighth USENIX Security Symposium*, August 1999.

[115] William H. Winsborough and Ninghui Li. Towards practical automated trust negotiation. In *Proceedings of the Third IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 92–103, June 2002.

[116] William H. Winsborough and Ninghui Li. Safety in automated trust negotiation. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 147–160, May 2004.

[117] William H. Winsborough, Kent E. Seamons, and Vicki E. Jones. Automated trust negotiation. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, January 2000.

[118] Marianne Winslett, Ting Yu, Kent E. Seamons, Adam Hess, Jared Jacobson, Ryan Jarvis, Bryan Smith, and Lina Yu. Negotiating trust on the Web. *IEEE Internet Computing*, 6(6):30–37, November/December 2002.

[119] Marianne Winslett, Charles C. Zhang, and Piero A. Bonatti. PeerAccess: A logic for distributed authorization. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS 2005)*, pages 168–179, November 2005.

[120] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE'03)*, pages 49–60, March 2003.

[121] Danfeng Yao, Keith Frikken, Mike Atallah, and Roberto Tamassia. Point-based trust: Define how much privacy is worth. In *Proceedings of the Eighth International Conference on Information and Communications Security (ICICS '06)*, number 4307 in Lecture Notes in Computer Science, pages 190–209. Springer, 2006.

[122] Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security*, 6(1):1–42, February 2003.

[123] Lidong Zhou, Fred B. Schneider, and Robbert van Renesse. COCA: A secure distributed online certification authority. *ACM Transactions on Computer Systems*, 20(4):329–368, November 2002.

[124] George K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.

Author's Biography

Adam J. Lee was born in Chester County, Pennsylvania in 1981 and grew up in Williamsburg, Virginia. In May 2003, he graduated *magna cum laude* from the College of Engineering at Cornell University with a B.S. in Computer Science and a minor in Applied Mathematics. Adam completed his graduate studies in Computer Science at the University of Illinois at Urbana-Champaign, receiving the M.S. and Ph.D. degrees in August 2005 and October 2008, respectively.

During the course of his graduate studies, Adam received several research honors including a Motorola Center for Communications Graduate Fellowship, a Cisco Systems Information Assurance Scholarship, and invitations to submit three papers to fully-refereed issues of *ACM Transactions on Information and System Security* containing extended versions of the top papers from the ACM CCS and SACMAT conferences. He also received teaching awards from the UIUC Department of Computer Science and the UIUC Center for Teaching Excellence.

Adam's research interests lie at the intersection of the computer security, privacy, and distributed systems fields. He is particularly interested in trust negotiation and distributed proof construction approaches to authorization, which can be used to facilitate secure interactions across multiple security domains while still preserving each individual's privacy and autonomy.