Authorization in Trust Management: Features and Foundations

Peter C. Chapin
University of Vermont
and
Christian Skalka
University of Vermont
and
X. Sean Wang
University of Vermont

Trust management systems are frameworks for authorization in modern distributed systems, allowing remotely accessible resources to be protected by providers. By allowing providers to specify policy, and access requesters to possess certain access rights, trust management automates the process of determining whether access should be allowed on the basis of policy, rights, and an authorization semantics. In this paper we survey modern state-of-the-art in trust management authorization, focusing on features of policy and rights languages that provide the necessary expressiveness for modern practice. We characterize systems in light of a generic structure that takes into account components of practical implementations. We emphasize systems that have a formal foundation, since security properties of them can be rigorously guaranteed. Underlying formalisms are reviewed to provide necessary background.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms: Security, Design, Languages

Additional Key Words and Phrases: Distributed Authorization, Trust Management Systems

1. INTRODUCTION

Distributed applications that span administrative domains have become commonplace in today's computing environment. Electronic commerce, high performance scientific computing, groupware, and multimedia applications all require collaborations between distinct social entities. In such systems each administrative domain, also called a security domain, controls access to its own resources and operates independently of other administrative domains. The problem of how to best specify and implement access control in such an environment has been a topic of considerable research. To address this problem the idea of *trust management* was introduced [Blaze et al. 1996] and subsequently developed by many authors, providing frameworks in which entities can specify independent access control policies that are enforced upon access request.

Authors' addresses: Peter Chapin, University of Vermont, Department of Computer Science, Burlington, VT 05405, pchapin@cs.uvm.edu. Christian Skalka, University of Vermont, Department of Computer Science, Burlington, VT 05405, skalka@cs.uvm.edu. X. Sean Wang, University of Vermont, Department of Computer Science, Burlington, VT 05405, xywang@cs.uvm.edu.

At the heart of trust management systems is the *authorization procedure*, which determines whether resource access should be granted or not based on a number of conditions. The semantics of authorization provide meaning to the features supported by trust management systems, for both the policy maker and the resource requester. While a number of techniques have been proposed to characterize authorization in trust management systems, we argue that the most promising are those based on rigorous formal foundations. This argument is not new, in fact it has motivated trust management research since its inception [Woo and Lam 1993]. In a security setting, entities should be able to specify policies precisely, to have an absolutely clear idea of the meaning of their policies, and to have confidence that they are correctly enforced by authorization mechanisms. Formally well-founded trust management systems achieve this, providing a setting in which reliability can be rigorously established by mathematical proof. In particular, various logics have served as the foundation for trust management [Abadi 2003; Bertino et al. 2003]. In this paper we survey state-of-the-art in trust management authorization, with an emphasis on formally well-founded systems. These systems are compared to each other with respect to desirable high-level features of trust management.

Our focus is the foundations and features of trust management systems, not their application, though we note that trust management systems have been shown to enforce security in many real applications. For example, the KeyNote system has been shown capable of enforcing the IPsec network protocol [Blaze et al. 2002; 2003]. SPKI/SDSI has been used to provide security in component based programming language design [Liu and Smith 2002]. Cassandra has been examined in the context of the United Kingdom's proposed nationwide electronic health records system [Becker and Sewell 2004b]. In addition, the Extensible Access Control Markup Language (XACML) [OASIS 2006a] and the Security Assertion Markup Language (SAML) [OASIS 2006b], both OASIS standards, define XML policy and assertion languages that makes use of many trust management concepts.

1.1 Authorization Frameworks

The trust management systems we survey are primarily concerned with *authorization*, as opposed to *authentication*. The latter addresses how to determine or verify the identity of actors or message signers in a distributed transaction with a high degree of confidence. Authorization, on the other hand, is based on calculi of principals whose identities are taken for granted. Although any real implementation of an authorization system will rely on authentication to establish these identities, and key-to-identity bindings may even have an abstract representation in the system, authorization generally treats authentication and public key infrastructure as orthogonal issues. Authorization is more properly concerned with non-trivial access control policies– how to specify them, what they mean, and how to endow trusted principals with the credentials necessary to satisfy them.

Authorization in trust management systems is more expressive than in traditional access control systems such as role based access control (RBAC) [Sandhu et al. 1996]. In such simpler models, access is based directly on identities of principals. But in a large distributed environment such as the Internet, creating a single local database of all potential requesters is untenable. Where there are multiple domains of administrative control, no single authorizer can be expected to have direct knowledge of all users of the system. Furthermore, the Internet is a highly dynamic and volatile environment, and no single entity can be expected to keep pace with changes in an authoritative manner. Finally, basing authorization purely on identity is not a sufficiently expressive or flexible approach, since

security in modern distributed systems utilizes more sophisticated features (e.g. delegation) and policies (e.g. separation of duty [Simon and Zurko 1997]). These problems are addressed by the use of trust management systems. We now return to some of the applications mentioned above, to illustrate how authorization in trust management systems is suited to enforcing security in practical computing scenarios.

IPsec. Blaze and Ioannidis [Blaze et al. 2002] describe an extension to the IPsec architecture that uses KeyNote to check if packet filters proposed by a remote host comply with a local policy for the creation of such filters. This allows a system administrator to prevent an attacker from negotiating a secure connection and then using that connection to attack vulnerable services. This application is an instance of the more general idea of using a trust management system for firewall management.

Web Page Content Ratings. Several authors describe the use of trust management systems to implement web page content rating schemes [Gunter et al. 1997; Chu et al. 1997]. This is of significant practical interest; the World Wide Web Consortium has considered using trust management concepts in its Platform for Internet Content Selection [Resnick and Miller 1996]. In a rating scheme a client delegates the authority to rate web pages to a suitable ratings server. The server issues certificates that bind a web page (via its hash value) to a rating. When a page is fetched, the web server delivers this certificate to the browser where the browser's policy is consulted to determine if the page should be displayed.

Medical Records. Several trust management systems have been applied to maintaining integrity and privacy in electronic health records [Bacon et al. 2002; Becker and Sewell 2004b], a topic of considerable importance in modern health care [Office of Technology Assessment 1993]. Security in this setting involves policies spanning many loosely coupled domains such as clinics, hospitals, laboratories, and emergency services.

1.2 Goals and Outline of the Paper

A summary and comparison of the features and formal underpinnings of authorization procedures in trust management systems is a primary goal of this paper, grounded in a review of their foundations in authorization logics such as ABLP [Abadi et al. 1993]. This summary provides a useful explanation and overview of modern state-of-the-art in trust management authorization technology. Another contribution of this survey is the characterization of authorization frameworks as *systems* that include other components in addition to the core authorization semantics. This distinguishes our presentation from a previous survey of authorization logics [Abadi 2003]. It is important to consider these components, since some features of trust management systems may be reflected in them rather than in the authorization semantics, for example certificate expiration dates may be checked when parsing wire format certificates but ignored by the authorization semantics. This also sheds light on how much formal support is provided for these features in various systems. We summarize the components of trust management systems, and compare them in light of which features are supported by which components.

Because trust management is a broad and active field, it is important to restrict the scope of our survey to provide sufficient depth as well as breadth. As the title suggests, we are mainly concerned with the semantics and implementation of *authorization* in trust managements systems, versus other components such as certificate storage and retrieval. We delineate our scope more precisely below in Sect. 2.1.

The remainder of this survey is organized as follows. In Sect. 2 we introduce important concepts and terminology, summarize the method we use to compare and contrast various systems, and introduce a running example. In Sect. 3 we highlight several features offered by trust management systems. Sect. 4 reviews in more detail the logical basis of trust management. Sect. 5 reviews several trust management systems with a focus on those that are logically well founded. Sect. 6 gives an overview of trust negotiation, an important component of some trust management applications. Finally we conclude in Sect. 7.

2. OVERVIEW

In this section we provide background in trust management systems for the general reader. We also clarify which trust management system components are relevant to the authorization decision– there turn out to be some important subtleties in this regard. In light of the structure of authorization decisions so described, we outline our approach to comparing trust management systems. We also provide a longer running example, which serves to illustrate the concepts introduced and later serves as an explicit point of comparison for the systems we survey.

2.1 Components of Full Implementations

Trust Management Systems (TMSs) in practice comprise a number of functions and subsystems, which we divide into three major components: *the authorization decision*, *certificate storage and retrieval*, and *trust negotiation*. Authorization decisions are relevant to the elements and semantics of the access control decision itself. Certificate storage and retrieval is relevant to the physical location of certificates that are the low-level representation of access control elements such as credentials and policies. For example, systems have been proposed for storing SPKI certificates using DNS [Nikander and Viljanen 1998] and for storing SDSI certificates using a peer-to-peer file server [Ajmani et al. 2002]. Trust negotiation [Winsborough et al. 2000; Yu et al. 2000; Seamons et al. 2001; Yu et al. 2001; Winsborough and Li 2002; 2004] is necessary for access control decisions where some elements of access policies or the credentials used to prove authorization with those polices should not be arbitrarily disclosed. For example, in [Winsborough et al. 2000] a scheme is proposed whereby access rights held by requesters are protected by their own policies, and both authorizers and requesters must show compliance with policies (i.e. negotiate) during authorization. We provide a brief summary and overview of trust negotation in Sect. 6, to provide a more complete view of trust management functionality and challenges in modern practice.

The importance of these other components notwithstanding, in this survey our focus will be on authorization decisions. This is because the authorization decision is the basis of any trust management system. Furthermore, not all the systems proposed in the literature have been developed sufficiently to include certificate storage implementations, nor trust negotiation strategies in the presence of confidentiality. Focusing on authorization decisions allows us to sufficiently narrow our scope, and thoroughly review components that endow systems with their characteristic features. When we say that we consider only those TMSs with a formal foundation in this survey as in Sect. 1, we mean that the authorization decision is based on a mathematically well-founded semantics of some sort, e.g. propositional logic or relational algebra.

2.2 Elements of Authorization: Glossary

To clarify the remaining presentation and identify fundamental elements of trust management authorization decisions, we now provide a glossary of relevant terms. More in depth discussion of these terms occurs throughout the rest of the paper, this section is intended as a succinct reference.

Entity: an individual actor in a distributed system, also frequently called a principal.

Resource: anything that a local system might regard as worthy of access control– file access, database lookup, web browser display area, etc.

Policy: a specification of rules for accessing a particular resource. Policy is usually defined locally at least in part, but TMSs sometimes allow policy to be defined non-locally as well.

Authorizer: the local authority that protects a resource, by automatically allowing access only after an appropriate proof of authorization has been shown. Authorizers also specify policy.

Requester: an entity (usually non-local) seeking to access a resource.

Attribute: a property of interest in some security domain, for example a role membership.

Credential: endows entities with certain attributes. Local policy usually specifies that requesters must be endowed with certain attributes before resource access is allowed, so credentials are essential to establish access rights to resources.

Issuer: the authority that issues a particular credential.

Certificate: a certified wire format representation of a credential.

Certificate revocation: the removal of a requester's credential, typically by the issuer.

Credential negation: Policy languages sometimes allow policy makers to specify that a credential *not* be held. Logically, this is expressed as credential negation.

Delegation of authority: the (usually temporary) logical transfer of authority over policy from one entity to another.

Delegation of rights: the (usually temporary) logical transfer of an access right from one entity to another.

Authorization decision: the determination of whether a given requester possesses the necessary attributes to access a particular resource as mediated by local policy, based on a preferably well-defined semantics of policies and credentials.

Authorization mechanism: the automated means by which an authorization decision is reached. Depending on context this refers to an algorithm or a module of software executed by the authorizer.

Core authorization semantics: the mathematically well-founded theory that constitutes the meaning of authorization decisions.

Role: an attribute that requesters can activate when requesting authorization. Authorization is often based on the role a requester is able to assume.

Role membership: an entity is said to be a member of a role if that entity is among the group of entities that can activate the role.

Threshold policy: threshold policies require a minimum specified number of entities to agree on some fact. Threshold policies usually support separation of duty authorization schemes [Li et al. 2002].

Domain: the security locality administered by a given authority.

Name space: the names defined in a particular domain.

2.3 Structure of an Authorization Decision

The subsystem of a trust management system that constitutes its authorization decision includes more than just a core authorization semantics. By *system* we mean the set of components that provide an implementation, not just an abstract specification of the authorization semantics. This distinguishes our presentation from a survey of authorization logics [Abadi 2003]. In this section we identify the components of a generic authorization decision and characterize its structure. This provides a better understanding of authorization decisions in general, and also a means to better categorize features of particular systems later in the paper.

In Fig. 1 we illustrate the components of a generic authorization decision. This graphic is meant as a rough sketch, not a formal specification, and not all TMSs contain all the components we describe. Nevertheless, the illustration is a useful tool for categorizing systems. The graphic is read top to bottom, and shows the flow of information through a particular authorization process, with output computed in response to an authorization request. The diagram is intentionally vague about the nature of the output: in the simplest case, the output is a simple "yes" or "no" decision as to whether or not to grant resource access, but in systems that support *trust negotiation*, the output could be a partial answer that provides direction for additional input. This issue is better discussed in Sect. 6. Within the scope of this survey, we mainly consider the case where the output is a boolean value, hence our terminology authorization *decision*. The core authorization semantics L implement the authorization decision, and may be a specialized inference system, or a proof search in a generic programming logic such as Prolog, for example. The authorization semantics takes as input parameters from C, P, and Q, which we now describe in detail.

Local policy P is defined in some specification language, that is transformed into terms understood by the core semantics by the transformation function T_P. This translation may just consist of parsing from concrete to abstract syntax, or T_P may compile statements in a high-level policy language into lower level terms for the core semantics. For example, TPL [Herzberg et al. 2000] provides an XML-based "trust policy language" that is compiled into Prolog.

Credentials for a particular requester may be defined as part of local policy. But an earmark of TMSs is their ability to extend local policies with credentials conferred by non-local authorities. This is realized as set of available certificates C that are transformed by a function T_C into credentials defined in terms understood by the core semantics. The transformation T_C provides a level of indirection allowing systems to choose between various certificate wire formats and PKIs, though X.509 [International Telecommunications Union 2000] or WS-Security [OASIS 2006c] are obvious choices for Internet and Web Services settings.

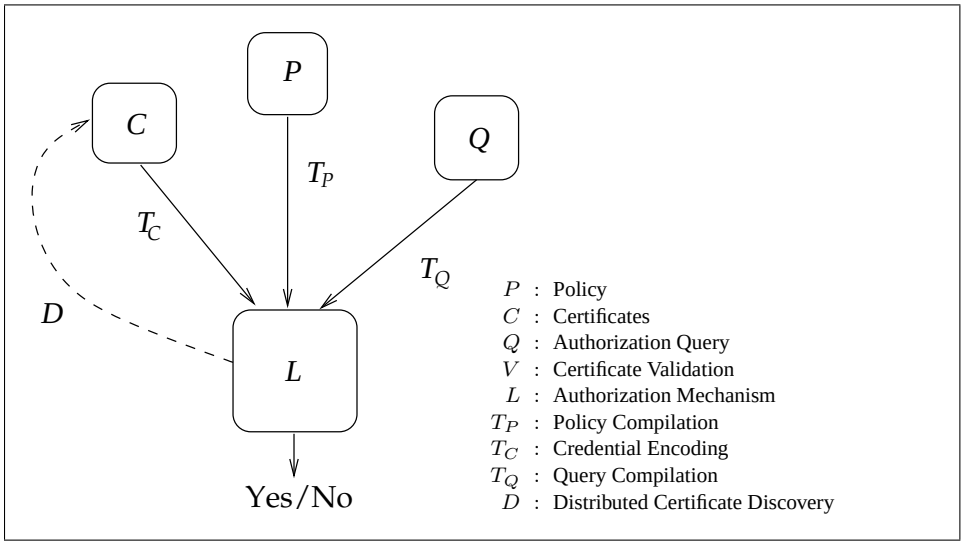

Fig. 1. Structure of an Authorization Decision

The transformation T_C also has special significance for the semantics of TMSs, since it is often not a straight parsing or compilation procedure. Rather, certificates may be rejected, or their credential representations enhanced, by certificate validity information. Validity information is external to the authorization semantics in some systems, but internal to it in others, so we represent the certificate validation component of the authorization decision V as dashed box. For example, any given certificate $c \in C$ almost always defines a finite lifetime for the certification, also called a validity interval [Winslett et al. 1997]. Some TMSs such as PCA [Bauer et al. 2002] support lifetime information in the authorization semantics, and in such a case T_C can map the lifetime information in c to its credential representation. However, other systems do not represent lifetimes in the authorization semantics per se (that is, in L), and in such cases the onus is on T_C to filter out expired certificates. For example, SPKI provides a mechanism for certificates to be checked online to see if they have been revoked [Ellison et al. 1999], but this mechanism is not part of SPKI's formal structure. This means on the one hand SPKI's revocation policy cannot be expressed in the SPKI policy language itself, nor enforced by its authorization semantics. On the other hand it allows a SPKI implementation to apply a different revocation policy without changing their underlying logical structure, and in general the difficulties associated with formalizing certificate revocation [Stubblebine 1995; Stubblebine and Wright 1996; Rivest 1998] can be avoided, while a means for certificate revocation in the system is still available.

In addition to policy P and certificates C, the authorization decision takes as input a question or goal Q that is specialized for a particular access request. As an example, some trust management systems, such as SDSI and RT_0 [Li et al. 2002; Li and Mitchell 2003b], define roles. These systems allow one to prove that a particular principal is in a particular role. Resources are associated with roles, and the authorization decision is based on whether the requester is a member of the relevant role. The transformation T_Q translates the goal into terms understood by the core semantics. Finally, the core semantics

combines policies and credentials established by input certificates to determine whether the authorization goal is satisfied, and outputs "yes" or "no" based on this determination.

However, as denoted by the dotted line, some systems also provide a "feedback" mechanism D between the semantics of authorization and certificate collection. Rather than merely answering "no" outright in case an authorization goal cannot be reached, the system might identify credentials that are missing and attempt to collect them. This functionality is sometimes called *distributed certificate chain discovery* [Li et al. 2003] or *policy directed certificate retrieval* [Gunter and Jim 2000b]. Whatever the specifics, it is clear that this functionality makes for a more flexible system in terms of certificate distribution and storage, but presents a significant challenge to system designers.

2.4 Comparing Trust Management Systems

A basis for comparing the features and functionality supported by trust management systems is fundamental to our survey. Since the systems we consider have a formal foundation, some sort of formal comparison seems appealing. Indeed, in [Weeks 2001] a framework is presented for describing a variety of authorization semantics, including KeyNote and SPKI/SDSI. This uniform specification of various semantics allows them to be compared on a completely formal basis, so for example it can be shown how credentials in one system can be faithfully encoded in another.

However, as we observe above, features of TMS authorization decisions are not entirely realized in the authorization semantics L, but may be realized in other components, as for example certificate revocation is sometimes implemented as part of the translation T_C from certificates to credentials. Since the definition of these components is not included in the formal specification of the authorization semantics, these system features can only be compared on an informal basis. We will therefore compare systems in light of the features they possess. In addition, we will observe whether the features are realized formally as part of the authorization semantics, or whether they are implemented by some other system component; this will clarify in what sense particular TMSs "possess" a certain feature. As a concrete point of comparison, we will also show how various systems encode the running example introduced next.

2.5 A Running Example

Suppose Alice is a cancer patient at a hospital being treated by Bob, a doctor. Alice grants Bob access to her medical records and also allows Bob to delegate such access to others as he sees fit.

Bob defines his team as a particular collection of individuals together with the people supporting them. A person supporting one of Bob's team members becomes a team member herself so Bob's definition is open ended and can potentially refer to a large number of people he does not know directly. Here we assume that Bob's team includes both medical and non-medical personnel (for example other doctors as well as receptionists). Bob then delegates his access to Alice's medical records to only the medical staff on his team– that is, people on his team that are also on the medical staff, as opposed to e.g. administrative staff.

Suppose further that Bob consults with another doctor, Carol, on Alice's condition. Bob modifies his policy to add Carol temporarily to his team. Carol orders some blood tests that are then analyzed by Dave, a lab technician and one of Carol's support people. The

policy described allows Dave to access Alice's medical records, for example to input the test results.

Dave signs the test results when he uploads them to the hospital database. He also includes appropriate credentials so that the database will authorize his access. The precise credentials needed depend on the trust management system in use and on the way credentials and policy statements are distributed and located by that system, however we imagine that these credentials should be able to express the following relations in some form:

—Bob has delegated his access to Alice's medical records to people on his team who are members of the medical staff.

—Carol is on Bob's team.

—If someone is on Bob's team, than any person on their support staff is also on Bob's team.

—Dave is one of Carol's support people.

—Dave is a member of the hospital's medical staff.

On the basis of these relations, one may deduce that Dave has access to Alice's medical records. Realistically, Dave may not know to submit all of this information, or have any knowledge of Bob's policy. If the trust management system used by the hospital supports distributed credential chain discovery, the hospital database would locate Bob's policy automatically in order to complete the authorization decision.

Complex access control scenarios such as this are difficult to express using traditional methods. Neither Alice nor Bob realize that Dave needs to be granted access to Alice's medical records. Although Dave's role as one of Carol's support people might be enough to grant him access to the records of Carol's patients, Dave's relationship to Bob, and hence to Alice, is indirect; it is Bob's act of adding Carol to his team that causes Dave to gain access to Alice's records. Observe also that Bob's team policy is recursive. A primary purpose of trust management systems is to provide language features and authorization semantics that support such complex policies.

3. FEATURES OF TRUST MANAGEMENT SYSTEMS

In this section we describe and discuss features relevant to trust management. We do not intend this listing to be exhaustive, rather we intend to focus on features that are generally considered important for trust management applications. Our goal is to more deeply characterize trust management systems, and to provide a means for comparison of various systems later in the paper.

We name and discuss features below, commenting on their relevance to trust management and noting important implementation issues. Particular trust management systems will be discussed in detail in Sect. 5, but in anticipation of that and in order to provide a thumbnail reference, Table I and Table II summarize feature sets for the collection of trust management systems we survey. Recall from the previous section that we understand trust management systems to include more than just the core authorization semantics, but also ancillary components such as translation from certificates to credentials. Thus, some systems are said to possess features that are realized in ancillary components instead of the core semantics. Also, while some systems are not explicitly designed to support certain features, their semantics is sufficiently expressive to simulate them, and such instances are listed in the table. The order in which the systems are listed is intended to follow an

approximate chronological order of their development. The order is approximate because some of the systems were developed over a considerable span of time and it is difficult to specify precisely when they reached a mature state.

3.1 Discussion of Features

We now briefly describe trust management features at a conceptual level. Specific examples of these features in systems are given in Sect. 5.

3.1.1 *Formal Foundation.* Since authorization systems are used in security-sensitive contexts, mathematically precise descriptions of their behavior and formal assurances of their correctness is essential. A variety of formalisms serve as effective foundations for the definition of trust management authorization semantics. As we describe later in this survey, these can be divided into three main categories: logics, database formalisms, and graph theory.

In the case of trust management systems based on logic, the authorization problem is expressed in terms of finding a proof of a particular formula representing successful resource access, with a collection of suitable axioms representing policy. Credentials relevant to a particular decision become additional hypotheses to be used in the proof. Trust management systems based on database formalisms (e.g. relational algebra) see the authorization decision as a query against a distributed database. The certificates issued by a principal contain, in effect, tuples from relations that a principal controls. Trust management systems based on graph theory define the authorization decision in terms of finding a path through a graph. The request is represented by a particular node in the graph. Principals are also graph nodes and the certificates they issue denote edges.

It is not unusual for a particular trust management system to be described by more than one formalism. In fact, some aspects of trust management are more naturally expressed using one formalism or another. Also, Datalog serves as both a database formalism and a programming logic, and several trust management systems have been specified in Datalog.

3.1.2 *Authorization Procedure. Authorization Complexity.* Trust management systems differ in exactly how the authorization decision is implemented. In a broad sense this is due to differences in the way the systems are described; systems using the same style of formalization tend to use similar authorization procedures. This is particularly evident among the systems using programming logics such as Datalog as both their formal foundation and implementation. However, some differences between systems result in significant differences in how authorization is computed even when the underlying formalism is the same, if certificate revocation is present in one system but not another for example. In some cases no authorization procedure is given; the details of computing authorization is entirely left to the implementors.

The computational complexity of the authorization decision is clearly of practical interest. Authorization should be decidable and tractable, but there is a trade off between the expressiveness of the certificate and policy language and the complexity of the authorization decision. For example, the systems that use Datalog with constraints (Datalog_C) can have various levels of computational complexity depending on the constraint domain used [Li and Mitchell 2003a]. Yet even trust management systems with undecidable decision procedures can be potentially useful; realistic policies may be decidable even if the general policy language is not.

	PolicyMaker	KeyNote	SPKI/SDSI	REFEREE	QCM	OASIS
Formal Foundation	Graph theory	Graph theory	Set theory. First order logic	Not formalized	Relational algebra	First order logic
Authorization Procedure	Graph search	Graph search	Tuple reduction	Arbitrary	Distributed database query	Unspecified
Authorization Complexity	Undecidable	Undecidable	Polynomial	Potentially undecidable	Unpublished	Undecidable
Public Key Infrastructure	External	External	External	Internal	Internal	External (undefined)
Threshold Policies	Yes	Yes	Yes	No	No	No
Local Name Spaces	Can be simulated	No	Yes	No	Yes	Yes
Role-Based Access Control	Can be simulated	Can be simulated	Yes	Can be simulated	Yes	Yes
Delegation of Rights	Yes	Yes	Yes (boolean depth)	Yes	No	Yes
Certificate Validity	Depends on assertion language	Internal	External	Internal	External	Internal
Credential Negation	No	No	No	Yes	No	No
Credential Revocation	External	External	On-line revocation check	External	Internal	Internal
Distributed Chain Discovery	No	No	No	Yes	Yes	No

Table I. Summary of Trust Management Systems, Part I

	PCA	TPL	SD3	Binder	RT	Cassandra	PROTUNE
Formal Foundation	Higher order logic	Prolog	Datalog	Datalog	Set theory. Datalog$_{\mathcal{C}}$	Datalog$_{\mathcal{C}}$	Stratified logic programs
Authorization Procedure	None	Not published	Datalog query	Datalog query	Graph search. Datalog query	Modified SLG resolution	Logic program evaluation
Authorization Complexity	Decidable proof checking	Undecidable	Polynomial	Polynomial	Polynomial	Depends on constraint domain	Potentially undecidable
Public Key Infrastructure	Internal	External	External	External	External	External	Internal
Threshold Policies	No	Can be simulated	No	No	Yes (RT^{T})	Yes	Can be simulated
Local Name Spaces	Can be simulated	No	Yes	Yes	Yes	Yes	Can be simulated
Role-Based Access Control	Can be simulated	Yes	Can be simulated	Can be simulated	Yes	Yes	Can be simulated
Delegation of Rights	Can be simulated	Yes (integer depth)	Can be simulated	Can be simulated	Yes (RT^{D})	Yes	Can be simulated
Certificate Validity	Internal	External	External	External	External	Internal	Internal
Credential Negation	No	Yes	No	No	No	No	Monotonic
Credential Revocation	Monotonic revocation	Internal	External	External	External	Internal	External
Distributed Chain Discovery	No	Yes (ad hoc)	Yes	No	Yes	Yes	Yes

Table II. Summary of Trust Management Systems, Part II

3.1.3 *Public Key Infrastructure (PKI).* It is common for trust management systems to treat keys directly as principals. This creates a conceptually clean design. In contrast some systems regard the human or machine participants as the principals and encode a relationship between principals and the keys that identify them. In the former case key bindings are not represented in the authorization semantics, where in the latter case they are. Although PKIs underpin the implementation of trust management systems, the question here is: to what extent does a particular trust management system directly concern itself with the details of key management.

3.1.4 *Threshold and Separation of Duty Policies.* Many systems support threshold policies, where at least k out of a set of n entities must agree on some point in order to grant access. Threshold policies are appealing since agreement provides confidence in situations wherein no single authority is trusted by itself. The concept of separation of duty is related to threshold policies. In the case of a separation of duty policy entities from different sets must agree before access is granted.

For example a bank might require that two different cashiers approve a withdrawal (same set—threshold policy). The bank might also require that a cashier and a manager, who are not the same person, approve a loan (different sets—separation of duty policy). In general threshold policies and separation of duty policies cannot be implemented in terms of each other, although some trust management systems provide support for both [Li et al. 2002].

3.1.5 *Local Name Spaces.* It is desirable for trust management systems to allow each administrative domain to manage its own name space independently. Requiring that names be globally unique is problematic and, in general infeasible. Although there have been attempts at creating a global name space [International Telecommunications Union 2001], these attempts have at best only been partially successful. The ability to reference non-local name spaces is also a keystone of modern trust management, in that it allows local policy to consider requesters that may not be directly known to the local system.

3.1.6 *Role-Based Access Control.* In a large system with many principals it is often convenient to use role based access control (RBAC) [Ferraiolo and Kuhn 1992; Sandhu et al. 1996]. In such a system *roles* are used to associate a group of principals to a set of permissions. The use of roles simplifies administration since the permissions granted to a potentially large group of principals are defined in a single place. RBAC is a conceptual foundation of modern authorization technologies, so many trust management systems provide features to support RBAC policies.

3.1.7 *Delegation of Rights.* All trust management systems allow an authorizer to delegate authority. In other words, an authorizer can specify third parties that have the authority to certify particular attributes. We take this as one of the defining characteristics of a trust management system. In many applications a requester will also want to delegate some or all of his or her rights to an intermediary who will act on that requester's behalf.

Delegation of rights is important in a distributed environment. For example a request may be made to an organization's front end system that accesses internal servers where the request is ultimately processed. The classic three-tier architecture of web applications follows this approach. In many environments the back end servers may have their own access control requirements, in which case the requester will need to delegate his or her rights to the front end system for use when making requests to the internal servers.

Trust management systems differ in their support for rights delegation. Delegation certificate forms may be formally provided, or delegation can be simulated via more primitive forms. Also, delegation *depth* can be modulated in some systems– rather than being purely transitive, delegation of rights may only be allowed to be transferred between fixed n principals. In some cases rights can be delegated arbitrarily or not at all. A system that has this latter feature is said to support boolean delegation depth.

3.1.8 *Certificate Validity.* Since an authorizer receives certificates from unknown and potentially untrustworthy entities, the validity of those certificates must be checked. Usually, signatures must be verified and the certificate must not have expired, since in practice certificates will almost always have a finite lifetime to ensure that obsolete information cannot circulate indefinitely. In some systems certificate validity is explicitly treated as part of the structure of the trust management authorization semantics– the component L described in Sect. 2.3. In such cases sufficient expressivity may exist in the policy language to specify authentication policies [Abadi et al. 1993], or, in a simpler (and currently more popular) scenario, certificate lifetimes can be directly represented in credentials and taken into account in policy [Bauer et al. 2002; Li and Feigenbaum 2002; Skalka et al. 2007]. In other systems, certificate validity is defined externally and checked as part of the translation of certificates into credentials– the component T_C– and not formally reflected in the authorization semantics [Ellison et al. 1999]. We note that it is a topic of lively debate whether authorizers [Rivest 1998] or certificate authorities [McDaniel and Rubin 2001] should determine validity intervals for authorization decisions.

3.1.9 *Credential Negation.* Policy languages sometimes allow policy makers to specify that a credential *not* be held. For example, access to a resource may require that requesters not possess a credential endowing them with a felon role. In systems using logic as a foundation for the semantics of authorization, this is expressed as credential negation. That is, authorization is predicated on the negation of a role attribute expressed as a credential. Note that this makes the semantics nonmonotonic– as more credentials (facts) are added to the system, it is possible that fewer authorizations succeed. As noted in [Seamons et al. 2002], this makes credential negation a generally undesirable feature, since nonmonotonic systems are potentially unsound in practice. For example, if a certificate is not discovered due to a network failure, access might be granted that would otherwise have been denied.

3.1.10 *Certificate Revocation.* Certificate revocation is similar to credential negation, but allows previously granted access rights to be explicitly eliminated [Rivest 1998]. Like certificate validity, this can be implemented in the translation T_C from certificates to credentials. For example, in SPKI/SDSI [Ellison et al. 1999] online revocation lists can be defined that filter out revoked certificates prior to embedding as credentials for the authorization decision. At first glance it may appear that certificate revocation entails nonmonotonicity, as does credential negation. However, it has been demonstrated that certificate revocation can encoded monotonically in both the Proof Carrying Authorization framework [Bauer et al. 2002] and a logic-based PKI infrastructure [Li and Feigenbaum 2002]; we describe how in Sect. 5.4.5. The technique points out a relation between certificate revocation and certificate validity, in that monotonic revocation can be based on lifetimes and the requirement to renew certificates. Various high-level approaches to and nuances of certificate revocation are discussed in [Rivest 1998].

3.1.11 *Distributed Certificate Chain Discovery.* Where do certificates for a particular access request come from? In the example in Sect. 2.5, it was assumed that the requester presents all relevant certificates upon access request. It is also easy to imagine settings in which authorizers maintain local databases of certificates. More generally, certificates could be stored anywhere in the network, as long as the local system has some way of finding them. Of course, given the potentially enormous number of certificates on the network, it is necessary to define some means of selectively retrieving only certificates that might pertain to a particular authorization decision. This problem is sometimes called *distributed certificate chain discovery* [Li et al. 2003] or *policy directed certificate retrieval* [Gunter and Jim 2000b]. In both of these approaches the process of obtaining certificates is formally well founded and not left to ad hoc techniques.

4. FOUNDATIONS OF AUTHORIZATION

Although trust management systems comprise a number of components as discussed in Sect. 2.3, the heart of any system is its authorization semantics, denoted L in Fig. 1, where the authorization decision is realized. The semantic foundations of authorization are well studied, having evolved from logical formalisms originally developed for verifying distributed authentication protocols [Burrows et al. 1990; Abadi et al. 1993], and early work on access control as a distinct concern in distributed systems [Woo and Lam 1993]. While formalisms other than logic have been used to specify authorization semantics, notably graph theory and relational algebra as discussed at various points in this paper, logic is probably the most popular, and logical approaches have had a broad impact on the foundations and practice of authorization and security [Bonatti and Samarati 2003]. Thus, a review of logical foundations provides historical perspective and insight into standard features and themes of authorization. To this end we now review BAN [Burrows et al. 1990] logic, the so-called logic of authentication [Abadi et al. 1993] which is commonly abbreviated ABLP, and aspects of programming logics such as Prolog and Datalog that are relevant to the issue.

4.1 BAN Authentication Logic

The first thorough study of a logic for specifying and verifying security protocols was presented in [Burrows et al. 1990] where a logic, commonly called BAN logic, was introduced. In that paper the authors analyze several authentication protocols, including Kerberos, Andrew Secure RPC, Needham-Schroeder Public-Key, and X.509. Although BAN was not intended as a foundation for authorization semantics, it is instructive to observe how it became so. From an authorization perspective, BAN is historically and technically significant because it introduces the ideas of representing beliefs, statements, and capabilities of participants in a distributed protocol using a formal logical framework.

BAN logic is a many-sorted modal logic that distinguishes between atomic principals P and encryption keys K. Formulae are created from propositional conjunction along with several additional constructs. These constructs include the following forms.

— P **believes** X: principal P might act as if statement X were true.

— P **sees** X: principal P has received a message containing statement X.

— P **said** X: at some point in the past (not necessarily during the current authentication session) principal P sent a message containing statement X.

— P **controls** X: principal P is an authority over X and should be trusted on it.

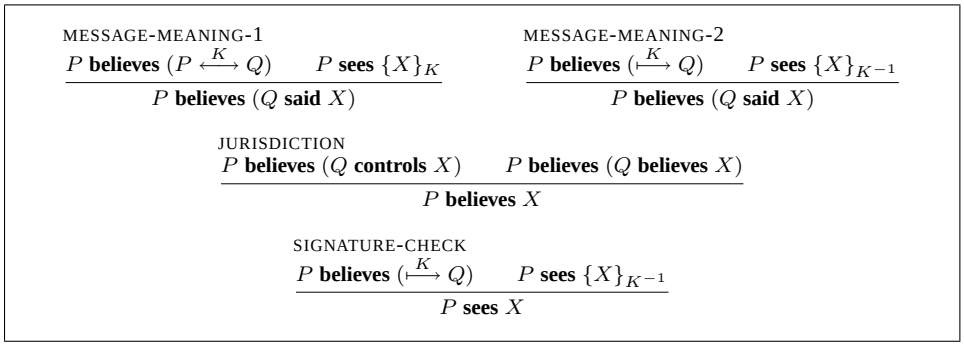

Fig. 2. Some Inference Rules of BAN Logic

— **fresh**(X): statement X is fresh. It has not been asserted during any previous authentication session.

— $P \overset{K}{\longleftrightarrow} Q$: principals P and Q can communicate using the shared key K.

— $\overset{K}{\longmapsto} P$: principal P has public key K. The private key corresponding to K is called K^{-1}.

— $\{X\}_K$: statement X is encrypted under key K. A statement encrypted under a private key is a signed statement.

BAN logic allows representation of statements a given principal says and believes as well as statements over which a principal has authority; these same ideas are used in authorization logics as well. In addition BAN logic allows one to talk about encryption keys, incorporating key security into the logic.

Inference rules formally specify the proof theory of BAN language constructs. A sampling of these inference rules is given in Fig. 2[1]. For example MESSAGE-MEANING-1 says that if P shares a key K with Q, and P receives a message encrypted with K, then P can conclude Q is the source of the message. The MESSAGE-MEANING-2 rule allows a similar inference for signed messages. These rules form the connection between the principals and the keys they use. Later authorization logics that consider interactions of principals and keys use similar rules to characterize these interactions.

The JURISDICTION rule formalizes a notion of delegation of authority, an essential ingredient in all trust management systems. The rule says that if P regards Q as an authority over X and P believes Q is asserting X, then P will accept Q's authority and believe X as well.

The SIGNATURE-CHECK rule encodes signature authentication and message encryption, allowing one to unwrap a signed message. If P believes key K is Q's public key and K successfully checks the signature on $\{X\}_{K^{-1}}$, then P sees the message content X. A similar rule exists for extracting the message content of a symmetrically encrypted message $\{X\}_K$. Inference rules that accept a message that P sees as a premise require that message to be signed or encrypted. Since the details of key signatures and encryption are generally hidden from the authorization component of most trust management systems as discussed in Sect. 1, these features of BAN make it more appropriate as a logic of authentication.

[1]The rule names in Fig. 2 and Fig. 3 have been made up by us to ease discussion.

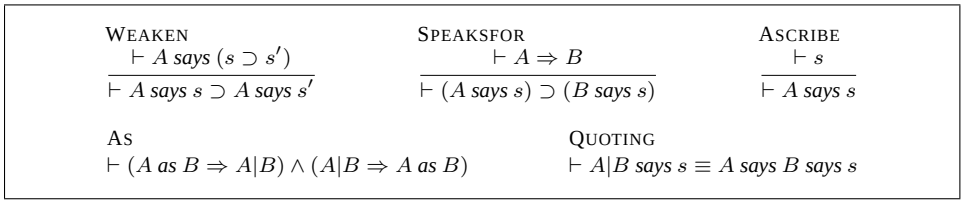

Fig. 3. Some Inference Rules and Axioms of ABLP Logic

As an example of representations in BAN logic, the statement:

$$A \textbf{ believes } (A \overset{K_{as}}{\longleftrightarrow} S)$$

expresses A's belief that she shares a key K_{as} with some authentication server S. The statement:

$$A \textbf{ sees } \{A \overset{K_{ab}}{\longleftrightarrow} B\}_{K_{as}}$$

represents a message sent to A containing a key intended to be shared between A and B and encrypted using a key shared between A and some authentication server S. The goal of the analysis might be to prove a formula such as

$$(A \textbf{ believes } (A \overset{K}{\longleftrightarrow} B)) \wedge (B \textbf{ believes } (A \overset{K}{\longleftrightarrow} B))$$

for some key K. Such a formula asserts that both A and B believe they have a key they share with each other.

4.2 ABLP Distributed Authorization Logic

The logical basis of many trust management systems is due to a general calculus for distributed access control called the logic of authentication and commonly abbreviated ABLP logic [Abadi et al. 1993]. (ABLP is an acronym for the author list of the seminal paper that develops the logic [Abadi et al. 1993]). ABLP develops many ideas introduced with BAN logic, but is intended less as a low-level specification language for authentication protocols, and more as a logic for reasoning about access control in general. As the authors discuss, this includes authorization issues– groups, roles, delegation, etc. Hence, ABLP logic formalizes a rich authorization semantics, and has inspired much subsequent development in trust management.

Full ABLP logic is sufficiently expressive to be undecidable. However, [Abadi et al. 1993] describes a number of restrictions to ABLP logic that allow for decidable access control decisions while still retaining enough expressivity to be useful. In practice, it has been used to support access control in the Taos operating system [Wobber et al. 1993].

In ABLP logic principals P can be users, roles, machines, I/O channels, encryption keys or any other convenient abstraction. In addition to atomic principals, denoted A, B, C, etc., *compound* principals can constructed with the use of connectives:

—$A \vee B$: this principal represents the group containing A and B.

—$A \wedge B$: this principal issues statements signed jointly by A and B.

—$A|B$: pronounced "A quoting B", this principal issues statements said by A to originate from B.

—A *for* B: this principal represents A speaking on behalf of B, which is a stronger notion than $A|B$.

—A *as* R: this principal represents A assuming the role R.

ABLP formulae s are then built from principals, standard logical connectives, and special connectives for representing authorization concepts:

—p, $\neg s$, $s \wedge s$, $s \supset s$: propositional atoms, negation, conjunction, and implication.

—$P_1 \Rightarrow P_2$: pronounced "P_1 speaks for P_2", this denotes that P_1 speaks with all the authority of P_2.

—P *says* s: this denotes that P has uttered the statement s.

In addition to the usual inference rules of propositional logic, inference rules and axioms are provided for the authorization-specific connectives, including those defined in Fig. 3. Notably, rule SPEAKSFOR says that if A speaks for B and A has asserted some s, then implicitly B has also asserted s. Rule AS establishes that the principal connective as can be defined as a derived form of (|). These rules together comprise a proof theory, where consequences can be deduced from assumptions.

In particular, resources can be represented as atomic propositions $\mathbf{priv}$, and access to the resource can be granted if the associated proposition can be proved given assumptions about policy and credentials. For example, with A *controls* s defined as syntactic sugar for $(A \textit{ says } s) \supset s$, access control lists may be modeled as conjunctions of assertions A *controls* $\mathbf{priv}$. An access request for $\mathbf{priv}$ by A is represented as the assumption $\vdash$ A *says* $\mathbf{priv}$. Observe that assumptions $\vdash$ A *says* s and $\vdash$ A *controls* s together imply $\vdash \mathbf{priv}$ by modus ponens, allowing access to the denoted resource.

The ABLP formula language can express a rich collection of authorization features. In addition to access control list encodings as described above, role membership can be modeled. To specify that A is a member of a role R, policy can include the assumption $\vdash A \Rightarrow R$. Then, when A assumes the role R to make an assertion s, represented as assumption $\vdash$ A *as* R *says* s, rules AS, SPEAKSFOR, and QUOTING allow deduction of $\vdash$ A *says* R *says* s, from which can be derived $\vdash$ R *says* R *says* s by SPEAKSFOR, hence $\vdash R|R$ *says* s by QUOTING, and finally R *says* s follows by assuming idempotence of (|) for roles [Abadi et al. 1993]. Note that ACL representations in this model need only take into account roles, e.g. R *controls* $\mathbf{priv}$ allows any specified member of R to gain access to $\mathbf{priv}$.

A variety of delegation idioms can also be modeled [Abadi et al. 1993]. An assertion of the form A *for* B *says* s means that A has asserted s on behalf of B, and denotes that B has delegated the assertion of s to A. In contrast, A *says* B *says* s merely represents A's claim that B asserts s, and requires no verification of the statement. The meaning of A *for* B can be altered according to desired delegation policies, for example A *for* B *says* s can be taken as syntactic sugar for the formula $A \wedge D$ *says* s, where D represents a delegation server.

4.3 Programming Logics

Programming logics such as Prolog and Datalog have played an important role in the development of trust management systems. As discussed above, logics provide useful abstractions for authorization semantics, furthermore specifications in executable programming

logics provide prototype implementations for free. Programming logics have served as target languages for the compilation of higher-level authorization languages [Li and Mitchell 2003a; Woo and Lam 1993], have served as the foundation for enriched authorization languages [Li and Mitchell 2006; Jim 2001; DeTreville 2002a; Li et al. 2002; Li et al. 2003], and have been used for the formalization and study of trust management systems [Li and Mitchell 2006; Polakow and Skalka 2006].

Both Prolog and Datalog are *Horn-Clause* logics, in which all formulae are restricted to the form *head* $\leftarrow$ *body*, where $\leftarrow$ is a right-to-left implication symbol, *head* is a proposition, and *body* is a conjunction of propositions. If variables X appear in a rule, the rule is implicitly universally quantified over those variables. The head of each rule is the consequent of the body. If *body* is empty then the rule is a *fact*.

As a simple example of how logics can apply in a trust management framework, imagine that delegation should be transitive. Suppose that $delegation(X, Y)$ is defined to mean that the rights of X have been delegated to Y. Suppose also that $cert(X, Y)$ represents a delegation certificate passing rights directly from X to Y. The following Horn clauses obtain transitivity of delegation:

$$delegation(X, Y) \leftarrow cert(X, Y)$$

$$delegation(X, Y) \leftarrow cert(X, Z),\ delegation(Z, Y)$$

Letting $a, b, c, ...$ denote constants, the following represents a collection of delegation certificates:

$$cert(a, b) \qquad cert(b, c) \qquad cert(b, d) \qquad cert(c, e)$$

From these facts and the definition of $delegation$, the query $delegation(a, e)$ will succeed while $delegation(d, e)$ fails.

Datalog was developed as a query language for databases. It is not a full programming language. In contrast Prolog is Turing complete and thus more expressive than Datalog. This extra expressivity is useful in certain contexts. For example, a full-featured authorization logic called Delegation Logic has been defined as a strict extension of Datalog at a high level, that is ultimately compiled to Prolog for practical implementation [Li et al. 2003]. However, Datalog has certain advantages in the authorization setting: the combination of monotonicity, a bottom-up proof strategy, and Datalog's *safety condition* (any variable appearing in the head of a rule must also appear in the body) guarantee program termination in polynomial time. In contrast, Prolog's top-down proof search can cause non-termination in the presence of cyclic dependencies. For example, if we added the certificate $cert(e, b)$ to the above fact set, some queries would not terminate. This problem is resolved by *tabling* as in XSB [XSB Inc. 2006], but it has been argued that this solution adds too much size and complexity to the implementation for authorization decisions [Li et al. 2002]. And while Datalog is not capable of expressing structured data, Datalog with constraints (Datalog$_C$), a restricted form of constraint logic programming [Jaffar and Maher 1994], has been shown sufficiently expressive for a wide range of trust management idioms [Li and Mitchell 2003a].

Prolog is able to express negation-as-failure, and so-called Disjunctive Datalog is likewise able to express a restricted form of negation [Eiter et al. 1997]. Therefore nonmonotonic authorization features such as credential negation can be provided in systems where programming logics are intended to serve as a basis for semantic interpretation or

implementation [Woo and Lam 1993; Bonatti and Samarati 2003]. However, as discussed in Sect. 3, nonmonotonicity in authorization semantics is generally considered undesirable, since it introduces the possibility of unsoundness in practice [Seamons et al. 2002]. Also, while certificate revocation seems at first blush to entail nonmonotonicity, it has been shown to be definable monotonically with appropriately constructed logical inference rules [Li and Feigenbaum 2002; Bauer et al. 2002]. Or, as discussed in Sect. 3, revocation can be handled by components external to the authorization semantics (via component T_C in Fig. 1), for example by filtering certificates through certificate revocation lists prior to authorization decisions as in SPKI/SDSI. For these reasons previous authors have argued that monotonic (subsets of) programming logics are adequate foundations for trust management applications, such as safe Datalog with constraint domains [Li and Mitchell 2003a].

Recently, more expressive programming logics have been proposed to address restrictions in the Horn-clause formula languages of Datalog and Prolog. Relevant work has proposed use of the higher-order linear logic programming language LolliMon as a foundation for trust management systems [Polakow and Skalka 2006]. LolliMon is not restricted to a Horn-clause form, and the availability of hypothetical (vs. strictly literal) subgoals and linear assumptions in particular allow the formal modeling of distributed certificate chain discovery (component D in Fig. 1), as interleaved with the authorization semantics of a trust management system.

5. REVIEW OF TRUST MANAGEMENT SYSTEMS

In this section we review a collection of trust management systems. We cover three systems in depth– SPKI/SDSI, QCM and its successor SD3, and RT– and more briefly summarize a number of others. We focus on SPKI/SDSI, RT, and QCM and SD3 because together they represent a fairly encompassing variety of approaches to trust management. Our entire review is not intended to be exhaustive, but rather representative of the breadth of trust management systems.

For each of the three systems we cover in depth, we begin by providing a summary overview of that system. We then describe the system's features, as enumerated in Table I and Table II, with an emphasis on those features that are unique to the system or otherwise worthy of attention. We then express the running example introduced in Sect. 2, in terms of the system's facilities. We follow this with a discussion of the semantics of the system's core logic and finally observations about system implementations.

5.1 SPKI/SDSI

The Simple Distributed Security Infrastructure (SDSI) [Rivest and Lampson 1996a; 1996b] is a system for managing distributed name spaces. In addition to global names, a primary contribution of SDSI is linked local name space management, where name spaces are defined and structured locally, but can be referenced non-locally. An authorizer associates access rights with a particular local name, and any principals *bound* to that name by SDSI *name certificates* are authorized for access. By signing access requests with names, an authorization logic is obtained based on name-to-key bindings, and linking relations between names.

The Simple Public Key Infrastructure (SPKI) was developed concurrently with SDSI to provide more complex authorization policies in distributed systems without the need for managing identities. These technologies merged into SPKI/SDSI version 2.0 [Ellison et al. 1999]. SPKI adds to SDSI the ability to directly bind a capability, called an *authorization*

certificate, to a name or key. Such bindings are only meaningful if the issuing principal has a superset of the capabilities being bound. Thus SPKI/SDSI allows a principal to explicitly delegate a subset of his or her rights to another principal or to a name representing a collection of principals, resulting in a rich authorization language.

5.1.1 *Features of SDSI.* In SDSI public keys have the status of principals, and there is no attempt to associate public keys with individual people, machines, or other entities, this being regarded as an external consideration. When discussing SPKI/SDSI we use the terms "principal" and "public key" interchangeably. The system implicitly assumes that private keys are secure and that statements signed by those keys reflect the intentions of the corresponding principal. Thus the SDSI authorization semantics does not concern itself with certificate signature checking, rather such checking is handled entirely by the processing of certificates prior to authorization (that is, by component T_C in Fig. 1).

In SDSI each principal defines a structured name space local to that principal by issuing *name certificates*, binding names to principals in a manner that confers the rights of the name to the principal. A SDSI name is of the form $K\ A$, where K is a key identifying a name space and A fully qualifies the name. Intuitively, we may read $K\ A$ as "K's A", i.e., a local name A in K's name space. In addition, SDSI provides *extended names* of the form $K\ A_1 \cdots A_n$ that allows linkage to non-local name spaces as discussed below. A name certificate is abstracted as a 4-tuple of the form (K, A, S, V) [Ellison et al. 1999] where:

—K is the principal issuing the certificate.

—A is the name in K's name space being defined.

—S is the subject of the certificate (the name being bound to A).

—V is certificate validity information.

Certificate subjects S can be other principals, names, or linked names. The certificate validity field V contains expiration times or information about where to obtain revocation or revalidation information on-line. An authorizer can use this information to check if a certificate has been revoked in real-time. A SDSI system disregards any certificates that are expired or have been revoked so validity information does not play a direct role in authorization decisions.

Since validity information V is not relevant to the SDSI authorization semantics, the following useful syntactic sugar can be defined for the consideration of SDSI authorization logic:

$$KA \rightarrow S \triangleq (K, A, S, V) \qquad V \text{ is valid}$$

Informally this means that the name A in K's name space is being defined as a local name for the subject S. For example, the certificate K_a `robert` $\rightarrow K_b$ indicates that K_b is bound to K_a's `robert`.

The meaning of a SDSI name is the set of principals bound to that name by valid name certificates. Using only the above certificate, the meaning of K_a's `robert` is the set $\{K_b\}$. SDSI specifically allows multiple name certificates to define the same name, so if K_a issued a second name certificate asserting K_a `robert` $\rightarrow K_{B'}$ then the meaning of K_a's `robert` would be the set $\{K_b, K_{B'}\}$. Thus SDSI names are essentially group names. When an authorizer associates access rights to a local name, that name behaves similarly to a role as defined by the role-based access control (RBAC) community [Ferraiolo and Kuhn 1992]. In this way SDSI provides support for RBAC.

An important aspect of SDSI is that it allows certificate subjects to refer to non-local names. This can be done via a certificate of the form $K_1\,A_1 \rightarrow K_2\,A_2$ denoting that the meaning of $K_1\,A_1$ subsumes that of $K_2\,A_2$, i.e. all names bound to $K_2\,A_2$ are also bound to $K_1\,A_1$. Certificate subjects can also be an extended name $K\,A_1\,\cdots\,A_n$. For $n = 2$, the extended name $K\,A_1\,A_2$ has a meaning that is based on the meaning of $K\,A_1$: it is the set of all names bound to $K_x\,A_2$ such that K_x is bound to $K\,A_1$. Iterating this idea obtains meaning for extended names with higher values of n.

5.1.2 *Running Example (SDSI).* Here we show how to encode the policies described in the medical records example in Sect. 2.5 using SDSI. Assuming that K_a, K_b, K_c, and K_d are Alice, Bob, Carol, and Dave's keys respectively, Alice's policy is expressed as:

—K_a `records` $\rightarrow K_b$
—K_a `records` $\rightarrow K_b$ `alice_delegates`

Although the original SDSI definitions provided a way to define groups of principals using set intersections, SDSI version 2.0 lacks this facility. This presents a problem in the example as originally stated since Bob would then be forced to grant his entire team, including non-medical personnel, access to Alice's medical records. To work around this Bob can distinguish between his overall team and his medical team, so his policy is:

—K_b `medical_team` $\rightarrow K_b$ `medical_team support`
—K_b `alice_delegates` $\rightarrow K_b$ `medical_team`
—K_b `medical_team` $\rightarrow K_c$

Carol's policy includes K_c `support` $\rightarrow K_d$ defining Dave as a member of her support staff. Again because SDSI 2.0 lacks intersections the hospital's assertion that Dave is a medical staff member is not used; we must presume that Carol will only add medical staff members to her support staff. In a more realistic situation Carol may want to distinguish between her medical support staff and her non-medical support staff by using two distinct names.

5.1.3 *Features of SPKI.* SPKI extends the SDSI framework with authorization certificates, allowing authorization rights to be delegated from principal to principal. Such certificates have the form of a 5-tuple, (K, S, D, T, V) where:

—K is the principal issuing the certificate.
—S is the subject of the certificate.
—D is a boolean delegation flag.
—T is the authorization tag.
—V is certificate validity information.

The K, S, and V fields are the same as for SDSI name certificates. The T field of the certificate is the *authorization tag*. It is formatted as an s-expression with specific rules regarding its structure. The meaning of the tag is left undefined by SPKI and is application specific. For example a tag such as `(http (port 8080) (read (url /downloads)))` might represent the capability of being able to read from the `downloads` directory on the HTTP server at port 8080. In this way, SPKI authorization tags provide a way to make statements about structured resources.

The D field of the certificate is the delegation flag. If set the subject is allowed to further delegate the authorization to others by issuing new authorization certificates as appropriate. SPKI provides only boolean delegation control where authorizations can be delegated arbitrarily or not at all. SPKI's design does not allow a principal to specify an integer delegation depth because of the inherent difficulty in specifying an appropriate depth. The argument for this is that in general principals can't easily know how many levels of delegation an authorization might reasonably need. Also since controlling the depth of delegation does not restrict the width of the delegation tree, a limited depth does not necessarily prevent rampant delegation [Ellison et al. 1999].

5.1.4 *Running Example (SPKI).* In the SDSI example above, a request signed by a key K is granted access to Alice's records if the name K_a `records` can be resolved to K. SPKI/SDSI version 2.0 contains SDSI and so this approach would apply in a SPKI/SDSI setting as well. However, SPKI also provides authorization certificates.

So far the example has treated Alice's medical records as a single entity. If access is granted to any part of Alice's records, access is granted to all of Alice's records. If Alice's medical records contain many components one could assert access control over each component individually using separate names to represent the different components. However, for indefinitely large structured resources such an approach is infeasible. SPKI authorization tags allow an indefinite subset of a structured resource to be specified and thus offers a granularity of control that is not possible using SDSI alone.

For example, Alice might issue a SPKI authorization certificate that grants Bob access to her medical records (or some portion thereof) and the power to delegate that access to others. Such a certificate might look like:

$$(K_a, K_b, \texttt{true}, \texttt{(records Alice (rw *))})$$

Here the authorization tag `(records Alice (rw *))` is assumed to convey full (read and write) access to all of Alice's records. Although the precise format of this string is application dependent, as long as the hospital database acting on behalf of Alice understands its meaning, authorizations will only be carried out according to Alice's wishes.

Bob could delegate the authorization he received from Alice to his medical team by issuing another authorization certificate:

$$(K_b, K_b\ \texttt{medical\_team}, \texttt{false}, \texttt{(records Alice (rw *))})$$

Here Bob prevents further delegations of the authorization. In this example, Bob passes his entire set of permissions to his medical team. Assuming Bob understands the format of the authorization tag, he could optionally pass a subset of his permissions to his team. When a request is made the hospital database would intersect the authorization tags to find the overall set of permissions allowed in the request.

In this example, no further authorization certificates are necessary. When Dave submits his test results to the hospital database, he must sign his request with his key and provide SDSI name certificates to prove his key's association with K_b `medical_team`. He must also provide the two authorization certificates showing that K_b `medical_team` is authorized to access Alice's medical records. Notice that one of these authorization certificates is signed by Alice and thus authority over her records ultimately comes from her.

5.1.5 *Semantics.* The original presentations of SPKI and SDSI [Rivest and Lampson 1996a; 1996b; Ellison et al. 1999] provide a thorough informal specification of its semantics and also sketch an operational meaning of certificates via rewrite rules as discussed below in Sect. 5.1.6, but a rigorous formal specification has been a distinct project carried out after the initial development of the system. The problem is surprisingly subtle, with a number of authors proposing alternate solutions.

The semantics proposed by Clarke et al. [Clarke et al. 2001] is constructed as the least solution of a system of containment constraints imposed by a given set of certificates. Let $\mathcal{K}$ be the set of principals mentioned in a given collection of name certificates $\mathcal{C}$. Let $\mathcal{N}_L$ be the set of local names of the form $K\ A$ where $K \in \mathcal{K}$, and A is one of the name identifiers mentioned in $\mathcal{C}$. Let $\mathcal{N}_E$ be the set of extended names of the form $K\ A_1\ A_2\ \ldots\ A_n$, $n \geq 2$ where $K \in \mathcal{K}$ and the $A_1, A_2, \ldots, A_n$ are all name identifiers mentioned in $\mathcal{C}$. Finally let $\mathcal{T} = \mathcal{K} \cup \mathcal{N}_L \cup \mathcal{N}_E$ be the set of all *terms* that can be formed using the principals and name identifiers in $\mathcal{C}$. Then the semantics of name spaces is defined via the *valuation function* $\mathcal{V} : \mathcal{T} \to \mathcal{P}(\mathcal{K})$ satisfying the equations:

$$\begin{aligned} \mathcal{V}(K) &= \{K\} \qquad \text{for all } K \in \mathcal{K} \\ \mathcal{V}(KA_1A_2 \ldots A_n) &= \bigcup_{K' \in \mathcal{V}(KA_1)} \mathcal{V}(K'A_2A_3 \ldots A_n) \end{aligned}$$

Furthermore, $\mathcal{V}$ is defined to be the least function satisfying the above equalities and the following system of inequalities:

$$\mathcal{V}(K\ A) \supseteq \mathcal{V}(S) \qquad (K, A, S, V) \in \mathcal{C}$$

This is a succinct and intuitive description, well-suited to modeling the meaning of name certificates.

In contrast to this approach, Abadi developed a logic of SDSI's linked local names [Abadi 1998]. The model theory of the logic plays a role similar to that of Clarke's semantics, but the proof theory is technically closer to the certificate rewrite rules proposed in RFC-2693 [Ellison et al. 1999], allowing characteristics of names such as associativity to be clarified, and relations between the rewrite rules and semantic model to be drawn more easily. In particular, Abadi shows the rewrite rules are sound with respect to the logic. However, Abadi's logic does allow conclusions about names to be drawn that would not be possible in Rivest and Lampson's Scheme.

Abadi uses the notation $n \mapsto v$ to indicate a binding (aka *mapping*) of name n to the value v in the name space of some *current principal*. Intuitively $n \mapsto v$ means "v speaks-for n": any statement asserted by v is implicitly a statement asserted by n, a relation similar to $\Rightarrow$ in ABLP logic (Sect. 4). Here values are terms consisting of local names, public keys aka *global names*, and extended names. Some bindings are published by the assumed current principal as signed certificates. For example a binding such as `alice` $\mapsto$ (`bob mother`) maps the current principal's local name `alice` to the extended name (`bob mother`). Abadi also extends the notion of mapping so that arbitrary principal expressions can map to other arbitrary principal expressions. For example $(n_1\ n_2) \mapsto K$ conveys that the extended name $n_1\ n_2$ is bound to the key K.

Abadi's logic for SDSI names is a modal logic with an obvious debt to ABLP logic. Formulae of the form p **says** s formalize certificates asserting proposition s that are signed by p. The standard axioms and rules of inference from propositional logic and modal logic are then extended to include the axioms in Fig. 4. Sets of certificates are represented as

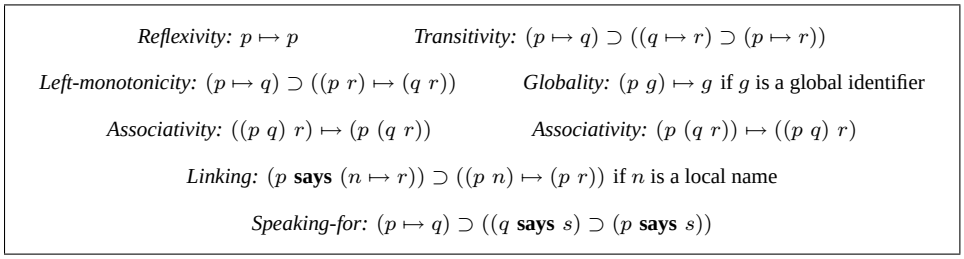

Fig. 4. Axioms of Abadi's Logic for SDSI Names

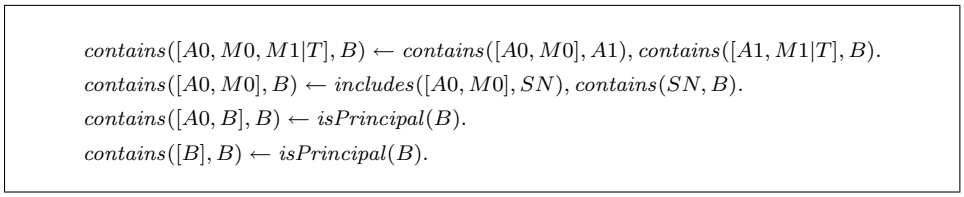

Fig. 5. Li's Logic Program for SDSI Name Resolution

logical assumptions, and name-to-key bindings $n \mapsto v$ are considered valid iff they can be deduced from these assumptions given the rules of inference. Abadi shows that the rules of deduction required to simulate the name resolution algorithm given by Rivest and Lampson are sound in this setting.

However, Abadi's general logic is more powerful than the name resolution rules allow. Consider the following example given by Abadi where f_1, f_2, and h are global names (keys). The assumptions are

$$m \mapsto f_1 \qquad m \mapsto f_2 \qquad f_1 \textbf{ says } (n_1 \mapsto n_2) \qquad f_2 \textbf{ says } (n_2 \mapsto h)$$

The logic allows one to deduce $(m\, n_1) \mapsto h$ whereas this result can not be obtained by the name resolution rules. Abadi suggests that such results may not be harmful.

Halpern and van der Meyden present an alternative to Abadi's logic [Halpern and van der Meyden 1999] that attempts to avoid some of the surprising conclusions in Abadi's logic while maintaining the correspondence between name resolution and proof. In the logic of Halpern and van der Meyden formulae of the form $p \mapsto q$ intuitively express the idea that all keys bound to q are also bound to p. This intuition is very similar to that expressed by the valuation function $\mathcal{V}$ described in [Clarke et al. 2001]. Halpern and van der Meyden avoid using "q speaks for p" as an intuitive explanation for $p \mapsto q$, regarding such a meaning as one about delegation and thus outside the scope of their study. Halpern and van der Meyden distinguish between general principal expressions and keys, restricting some of Abadi's axioms to operate only over keys rather than general principal expressions. These authors later accounted for additional features in SPKI/SDSI [Halpern and van der Meyden 2001], including authorization certificates and certificate lifetime and revocation issues. Howell and Kotz also provide a logical accounting of SPKI/SDSI [Howell and Kotz 2000; Howell 2000] building on Abadi's concepts for SDSI names but with a restricted speaks-for relation.

Li provides yet another formulation of the logic of SDSI local names [Li 2000], but based on general purpose programming logics rather than special purpose authorization

logics. Li regards the handling of $\mapsto$ in Abadi as too general and observes that even under the more restricted axioms of Halpern and van der Meyden there are some undesirable consequences. Instead Li presents the Prolog logic program in Fig. 5 that performs SDSI name resolution. In this program an extended SDSI name $(am_1m_2 \ldots m_k)$ is represented by a list $[a, m_1, m_2, \ldots, m_k]$ [2]. Name certificates are translated into facts using the $includes$ predicate, such that $K\ A \rightarrow S$ becomes $includes([K, A], [S])$. In related work Li and Mitchell show the equivalence of a logic programming semantics for SPKI/SDSI and a set-theory semantics in the style of Clarke [Li and Mitchell 2006].

Jha and Reps describe a connection between SPKI/SDSI and pushdown systems [Jha and Reps 2002], and show how to use model checking techniques to compute a proof of authorization. Existing model checking algorithms allow a variety of other questions to be answered as well, for example given a resource one might ask what names are able to access that resource.

5.1.6 *Implementation.* RFC-2693 [Ellison et al. 1999] defines a 4-tuple reduction rule that can be used to combine two related name certificates into a third certificate. This rule involves replacing a local name in one certificate with a key binding established by another. So, letting $\circ$ be an infix denotation of the rewrite operation, we have that:

$$K_1\ A \rightarrow K_2\ B_1\ B_2\ B_3 \circ K_2\ B_1 \rightarrow K_3$$

results in:

$$K_1\ A \rightarrow K_3\ B_2\ B_3$$

A similar reduction is defined for authorization certificates, describing how an authorization is explicitly delegated from one subject to the next. For example:

$$(K_1, S_1, \texttt{true}, T_1, V_1) \circ (S_1, S_2, D_2, T_2, V_2)$$

results in:

$$(K_1, S_2, D_2, A_I(T_1, T_2), V_I(V_1, V_2))$$

where A_I computes the intersection of the two authorizations and V_I computes the intersection of the two validity conditions. Rules for computing these intersections are given in RFC-2693. Finally a similar reduction rule describes how the subject of an authorization can be rewritten according to bindings specified in a name certificate.

Note that RFC-2693 does not give a specific algorithm for finding which reductions should be used, given a set of certificates for a particular access request. Instead the requester is required to send the appropriate certificates in the correct order. This puts the burden of constructing the proof of authorization on the requester; the authorizer merely checks this proof. This general concept is extended in Proof Carrying Authorization which we discuss in more detail in Sect. 5.4.5.

To relieve the access requester of the burden of proof, Clarke et al. describe a credential chain discovery algorithm, that will automatically check for authorization given a set of certificates and a particular request [Clarke et al. 2001]. The algorithm uses a graph construction to search for a particular sequence of reductions to a certificate delegating the requested permission from the local name space to the requester. This algorithm runs in

[2] Note that the Prolog syntax $[X, Y|T]$ represents a list whose first and second elements are X and Y respectively, and T is the rest (the tail) of the list.

$O(n^3L)$ time (worst case) where n is the number of input certificates and L is the length of the longest extended name in any of the certificates.

In both Clarke et al.'s scheme and that proposed by RFC-2693, it is assumed that all certificates are on hand when the authorization decision is made. To our knowledge no method for retrieving non-local SPKI/SDSI certificates dynamically has been described in the literature, in other words there are no distributed certificate chain discovery techniques developed for SPKI/SDSI in the sense described in Sect. 2.3.

5.2 RT

The RT trust management framework is not a single trust management system but rather a collection of trust management systems with varying expressiveness and complexity [Li et al. 2002; Li et al. 2003; Li and Mitchell 2003b]. The base system, RT_0, is similar to SDSI except that it limits extended names to one level of indirection and provides intersection roles. The limitation of linked roles to one level of indirection does not reduce the expressiveness of the language since additional indirections are possible by introducing intermediate roles.

RT_1 is an extension of RT_0 providing parameterized roles. RT_1^C further extends RT_1 to allow for the description of structured resources [Li and Mitchell 2003a; 2003b]. The system RT^D provides a mechanism to describe the delegation of rights and role activations, and RT^T provides support for threshold and separation of duty policies. RT^T and RT^D can be used in combination with RT_0, RT_1, or RT_1^C to create trust management systems such as RT_0^T, RT_1^{TD}, and so forth. A rich complexity analysis has also been developed for the RT framework for problems beyond simple authorization, e.g. role inclusion and role membership bounds [Li et al. 2005].

5.2.1 *Features.* Like SPKI/SDSI, the RT framework represents principals as public keys and does not attempt to formalize the connection between a key and an individual. The RT literature usually refers to these principals as *entities*. Also like SPKI/SDSI, the RT framework allows each entity to define roles in a name space that is local to that entity. An authorizer associates permissions with a particular role; to access a resource a requester must prove membership in the role. In this way the RT framework provides role based access control.

To define a role, an entity issues credentials that specify the role's membership. Some of these credentials may be a part of private policy, others may be signed by the issuer and made publicly available as certificates. The overall membership of a role is taken as the union of the memberships specified by all the defining credentials.

Let $A, B, C, \ldots$ range over entities and let $r, s, t, \ldots$ range over role names. A role r local to an entity A is denoted by $A.r$. RT_0 credentials are of the form $A.r \longleftarrow f$, where f can take on one of four forms to obtain one of four credential types:

(1) $A.r \longleftarrow E$
This form asserts that entity E is a member of role $A.r$.

(2) $A.r \longleftarrow B.s$
This form asserts that all members of role $B.s$ are members of role $A.r$. Credentials of this form can be used to delegate authority over the membership of a role to another entity.

(3) $A.r \longleftarrow B.s.t$

This form asserts that for each member E of $B.s$, all members of role $E.t$ are members of role $A.r$. Credentials of this form can be used to delegate authority over the membership of a role to all entities that have the attribute represented by $B.s$. The expression $B.s.t$ is called a *linked role*.

(4) $A.r \longleftarrow f_1 \cap \cdots \cap f_n$
This form asserts that each entity that is a member of all roles $f_1, \ldots, f_n$ is also a member of role $A.r$. The expression $f_1 \cap \cdots \cap f_n$ is called an *intersection role*.

For all credential forms $A.r \longleftarrow f$, the principal A is called the *issuer* of the credential.

RT_1 enhances RT_0 by allowing roles to be parameterized. For example, the second credential form above is extended to $A.r(h_1, h_2, \ldots, h_n) \longleftarrow B.s(k_1, k_2, \ldots, k_m)$ where the h_i and k_j are parameters. Role parameters are typed and can be integers, floating point values, dates and times, enumerations, or finite sets or ranges of these datatypes. An RT_1 credential is *well formed* if the parameters given to the roles have the right type and if each variable in the credential appears in the body of that credential.

As an example of an RT_1 credential [Li et al. 2002], suppose company A has a policy that the manager of an entity also evaluates that entity. This can be expressed in RT_1 using a policy statement such as

$$A.evaluatorOf(?Y) \longleftarrow A.managerOf(?Y)$$

This policy can't be feasibly expressed in RT_0 because the role parameters might take on an arbitrarily large number of values. In RT_0 individual credentials would be needed for each possible value of the role parameter.

RT_1^C further enhances the expressive power of RT_1 by allowing structured constraints to be applied to role parameters. In addition the restriction on variables only appearing in the body of a rule is lifted [Li and Mitchell 2003a; 2003b]. For example, suppose a host H wishes to grant access to a particular range of TCP ports to those entities that are employed by the information technology department. The host might have as its local policy:

$$Host.p(port \in [1024..2048]) \longleftarrow IT.employee$$

This example assumes that an entity is granted access to a particular TCP port if that entity is a member of the $Host.p$ role with the port specified as a parameter.

To accommodate threshold structures, representing agreement between a group of principals, the system RT^T interprets roles as sets of sets of entities, called *principal sets*. These principle sets can be combined with role product operators $\odot$ and $\otimes$. The features introduced by RT^T allow threshold policies and separation of duty policies to be written [Li et al. 2002].

RT^D adds the concepts of role activations and delegations to RT_0, via the delegation credential form $A \stackrel{C\ as\ D.r}{\longrightarrow} B$. In this case A delegates to B the *role activation* of $C\ as\ D.r$. Empowered with this role activation B can then access whatever facilities C can access from role $D.r$. This presupposes that A has been delegated the activation $C\ as\ D.r$, which holds when $A = C$ and A is a member of role $D.r$ in the basic case. Hence, delegated activations don't carry any authority unless there is a chain of delegation credentials where the credential at the head of the chain was issued by the entity mentioned in the role activation.

While the original RT framework does not support revocation in its policy language, it is proposed to incorporate revocation [Li et al. 2002] by leveraging a monotonic approach developed in [Li and Feigenbaum 2002] based on certificate lifetimes. While lifetimes and

the requirement for freshness are encoded logically, the proposal suggests the use of external certificate revocation lists to implement verification; this is an interesting example of the possible interplay between the semantics of authorization per se and components external to them. In addition, a variant of the RT framework has been developed that associates risk values with credentials [Skalka et al. 2007]. These risks are tracked through the authorization process so that the role membership is parameterized by the total membership risk. The set of risks and their ordering is left abstract, and can be specialized to a number of applications, e.g. risk can be defined as remaining certificate lifetime, so that role membership is parameterized by the minimal lifetime of certificates used for authorization.

5.2.2 *Running Example.* To express the medical records example using RT, only the facilities of RT_0 are necessary. Alice defines a role `records` whose members are able to access her medical records. She creates the policy

—`Alice.records` ← `Bob`
—`Alice.records` ← `Bob.alice_delegates`

The first rule grants her doctor, Bob, access to her records. The second rule allows Bob to further delegate that access by defining the membership of an `alice_delegates` role.

Bob's standing policy is

—`Bob.team` ← `Bob.team.support`
—`Bob.alice_delegates` ← `Hospital.medical_staff` ∩ `Bob.team`

The first rule defines Bob's team as including all the support personnel specified by the members of his team. In the second rule, Bob uses an intersection role to specify that only the medical personnel on his team should have access to Alice's medical records.

When Bob consults with Carol he adds `Bob.team` ← `Carol` to his policy to add Carol, and indirectly all of Carol's support people, to his team.

The only part of Carol's policy relevant to this example places Dave in her `support` role: `Carol.support` ← `Dave`. Finally Dave has a credential from the hospital asserting his membership in the `medical_staff` role. RT_0 can use these credentials to prove that Dave is a member of `Alice.records` and thus able to access Alice's medical records.

5.2.3 *Semantics.* The original formal semantics of RT is based on Datalog [Li et al. 2002]. Specifically each RT credential is translated into a Datalog rule. The meaning of a collection of RT credentials is defined in terms of the minimum model of the corresponding Datalog program. In the case of the RT_1^C, Datalog with constraints is used [Li and Mitchell 2003a].

The translation from RT_0 to Datalog requires only a single predicate *isMember* to assert when a particular entity is a member of a particular role. The translation rules are shown below where Datalog variables are shown prefixed with '*?*' to distinguish them from constants.

(1) $A.r \longleftarrow E$
isMember(E, A, r).

(2) $A.r \longleftarrow B.s$
isMember$(?x, A, r)$ ← *isMember*$(?x, B, s)$.

(3) $A.r \longleftarrow B.s.t$
isMember(*?x*, A, r) $\leftarrow$ *isMember*(*?y*, B, s), *isMember*(*?x*, *?y*, t).

(4) $A.r \longleftarrow B_1.s_1 \cap \cdots \cap B_n.s_n$
isMember(*?x*, A, r) $\leftarrow$ *isMember*(*?x*, B_1, s_1), ..., *isMember*(*?x*, B_n, s_n).

The authorizer associates a permission with a particular role, say $A.g$, called the *governing role*. Access is granted to an entity E iff the Datalog query *isMember*(E, A, g) succeeds.

An alternative set-theory semantics has also been defined for RT_0 [Li et al. 2003]. In this semantics each role $A.r$ is represented as a set of entities rmem$(A.r)$ that are members of that role. For a given set of credentials $\mathcal{C}$ these sets are the least sets satisfying the set of inequalities

$$\{\text{rmem}(A.r) \supseteq \text{expr[rmem]}(e) \mid A.r \longleftarrow e \in \mathcal{C}\}$$

where expr[rmem](e) is the set of entities in a particular role expression e. A *role expression* includes both linked roles and intersection roles. In particular:

$$\begin{aligned}
\text{expr[rmem]}(B) &= \{B\} \\
\text{expr[rmem]}(A.r) &= \text{rmem}(A.r) \\
\text{expr[rmem]}(A.r_1.r_2) &= \bigcup_{B \in \text{rmem}(A.r_1)} \text{rmem}(B.r_2) \\
\text{expr[rmem]}(f_1 \cap \cdots \cap f_k) &= \bigcap_{1 \leq j \leq k} \text{expr[rmem]}(f_j)
\end{aligned}$$

The set-theory semantics for RT_0 was developed primarily to provide theoretical support for a distributed credential chain discovery algorithm [Li et al. 2003]. The set-theory semantics facilitate proving soundness and completeness of that algorithm.

Another approach to the semantic specification of RT is taken by Polakow and Skalka, who propose the LolliMon linear logic programming language as a foundation [Polakow and Skalka 2006]. Like the set-theoretic specification, this approach has the advantage of being easily extended to the problem of distributed certificate chain discovery, while enjoying the additional benefit of scalability to the full RT framework. The encoding closely resembles the original Datalog *isMember* predicate defined above, and the logic of certificate discovery can be expressed by additional clauses in LolliMon's rich formula language.

5.2.4 *Implementation.* Li et al. describe an implementation strategy for RT_0 in terms of a construct called a credential graph $\mathcal{G}_\mathcal{C}$ [Li et al. 2003]. Each node in $\mathcal{G}_\mathcal{C}$ represents a role expression with directed edges corresponding to each credential. In addition, *derived edges* are added to represent the indirect relationships between roles that are introduced by linked roles and intersections. An entity is a member of a role iff there exists a path from the entity to the role in $\mathcal{G}_\mathcal{C}$. Li et al. prove that credential graphs are sound and complete with respect to the set-theory semantics of RT_0.

In addition Li et al. describe a distributed credential chain discovery algorithm that finds a path in $\mathcal{G}_\mathcal{C}$ given initially incomplete credentials [Li et al. 2003]. The algorithm assumes that either the issuer or subject of a credential can be contacted on-line and queried for more credentials on demand. In this way missing credentials can be found as needed to complete a proof of authorization. The algorithm can work either backward, starting at the governing role and following credentials from issuer to subject, or forward, starting at the entity representing the requester and following credentials from subject to issuer. In general

both approaches are useful. In some cases a certificate authority will maintain a database of all credentials issued, making the backward discovery algorithm effective. In other cases credentials will be held by the subjects, making the forward discovery algorithm more appropriate. To ensure that searches always succeed when possible, a type system can be used to assign appropriate types to role names. These types restrict the way credentials can be formed and specify where credentials must be stored [Li et al. 2003].

The complexity of credential chain discovery in RT_0 has been shown to be log-space $\mathcal{P}$-complete using a reduction from the monotone circuit value problem [Li et al. 2003].

5.3 QCM and SD3

Many trust management systems have focused on authorization decisions while setting aside issues of certificate storage and retrieval. In contrast, Query Certificate Manager (QCM) [Gunter and Jim 1997; Gunter et al. 1997; Gunter and Jim 2000b] and its successor Secure Dynamically Distributed Datalog (SD3) [Jim 2001; Jim and Suciu 2001] address the issue head-on, by treating trust management as essentially a distributed database problem. An advantage of their approach is that well-studied database techniques and abstractions can be leveraged. In particular, the system provides applications programmers with high-level database query languages for defining authorization policy over a transparent PKI infrastructure, where authorization is implemented as a query processed automatically over a distributed database. Among the implementation benefits of their distributed database approach are a variety of optimization techniques and a natural incorporation of distributed certificate retrieval. SD3 also introduces a novel certified evaluation mechanism that reduces the size of its trusted computing base.

5.3.1 *Features.* QCM hides from end-users the complexity of distributed query evaluation and certificate retrieval. Instead it presents a high level abstraction of a secure, local database. The original presentation of QCM proposed a policy language based on relational algebra [Gunter and Jim 1997]. Consider the following example of a web page content filtering application taken from Gunter and Jim [Gunter and Jim 1997]. Here a ratings server r_1 maintains a relation **ratings** containing rating information for a large collection of web pages. An independent server r_2 maintains a similar relation. A web browser then defines a local relation in terms of these other two using relational algebra expressions:

$$\begin{aligned} \textbf{ratings} &= r_1.\textbf{ratings} \cap r_2.\textbf{ratings} \\ \textbf{ok} &= \pi_{\text{hash}}(\sigma_{\text{rating}=G}\textbf{ratings}) \end{aligned}$$

The browser will only accept ratings for which the two rating servers agree. In addition, the browser's **ok** table contains only the page hashes for the pages with a G rating.

Now, suppose that the browser was governed by the policy that it would only display web pages with a G rating. Formally, assuming that h is the hash of a web page, it would only be displayed if $\sigma_{\text{hash}=h}\textbf{ok}$ is not null. In the simplest scenario, the browser could enforce this by submitting the query $\sigma_{\text{hash}=h}\textbf{ok}$ to a local QCM processor, and the processor would in turn query r_1 and r_2 remotely. However, a more efficient and flexible scheme is allowed in QCM– tuples in a database relation can be certified by the relation authority, and distributed as certificates. Such certificates, called *inclusions*, are denoted via an ABLP-style **says** connective. For example, the web server that hosts a page with hash h can obtain certificates from the rating servers for that page. Each certificate is signed by the

corresponding rating server:

$$r_1 \textbf{ says } r_1.\textbf{ratings} \supseteq \{(\text{hash} : h, \text{rating} : G)\}$$
$$r_2 \textbf{ says } r_2.\textbf{ratings} \supseteq \{(\text{hash} : h, \text{rating} : G)\}$$

When the page is requested by a browser, the two certificates are sent as well, and from there to the local QCM processor along with the query $\sigma_{\text{hash}=h}\textbf{ok}$. Now the processor does not need to contact the remote servers because the certificates contain enough information to answer the query directly, and can even cache the certificate contents for future use. This scheme is clearly more efficient, and has the additional benefit that not all relation authorities need to be online during authorization.

Also notable is QCM's support for certificate revocation [Gunter and Jim 2000a]. This is done by allowing a set of revoked tuples to be subtracted from a set of otherwise potentially useful tuples. QCM provides for explicit *non-membership certificates* that can be used to assert that a tuple is not an element of the revoked set. This later work adopts a set-theoretic model of the QCM database to accommodate the notion of non-membership, and a new language of set comprehension is defined for QCM programming.

As an example of this set comprehension language, consider the earlier example of web page content filtering. Rating server r_1 maintains a relation that binds page hashes to rating values $r_1.\textbf{ratings}(\text{hash}, \text{rating})$, and similarly for rating server r_2. The browser's policy then defines a set **ok** of acceptable page hashes with the statement

$$\begin{aligned}\textbf{ok} = \{h \mid \langle \text{hash} = h, \text{rating} = G\rangle &\leftarrow r_1.\textbf{ratings},\\ \langle \text{hash} = h, \text{rating} = G\rangle &\leftarrow r_2.\textbf{ratings}\,\}\end{aligned}$$

This defines a set of hash values for pages that both ratings servers agree are G rated. Queries also have the form of set expressions. When the browser retrieves a page it asks its local QCM processor to evaluate the query $\{p \mid p \leftarrow \textbf{ok}, p = h\}$ where h is the hash of the retrieved page. If this set evaluates to the singleton $\{h\}$ then the browser can display the page as G rated.

Secure Dynamically Distributed Datalog (SD3) is the successor of QCM. It adds to QCM an extended version of Datalog as its policy and credential language, allowing recursive policies to be defined. In SD3 predicates are scoped by public keys; rules can refer to predicates in other name spaces by prepending the key to the predicate name. For example, suppose that the predicate E under control of key K defines the edge relation of a particular graph. The following SD3 program computes the transitive closure of that graph.

$$\begin{aligned}T(X,Y) &\leftarrow K\$E(X,Y).\\ T(X,Y) &\leftarrow T(X,Z), T(Z,Y).\end{aligned}$$

SD3 also adds other notable implementation features, including intentional responses and certified evaluation, discussed below.

5.3.2 *Running Example.* Here we demonstrate how SD3 would express the medical records example. We need to first express the information to be processed as a collection of relations. Alice maintains a one-place relation `records` with tuples storing the keys of those principals who can access her medical records. We can then represent Alice's policy as the following two SD3-style Datalog rules.

—K_a`$records(`K_b`)`
—K_a`$records(X)` $\leftarrow$ K_b`$alice_delegates(X)`

Datalog has no problems expressing either recursion or intersections (conjunctions). Bob's policy becomes

—K_b`$team(X)` $\leftarrow$ K_b`$team(Y), Y$support(X)`
—K_b`$alice_delegates(X)` $\leftarrow$ K_h`$medical_staff(X),` K_b`$team(X)`
—K_b`$team(`K_c`)`

The remaining assertions made by the hospital and by Carol are

—K_h`$medical_staff(`K_d`)`
—K_c`$support(`K_d`)`

This example only hints at the expressivity of the general SD3 policy language.

SD3 distinguishes between global names that are key-qualified and local names that are not. In this way SD3 supports multiple, independent name spaces. If the hospital database evaluates Dave's request to update Alice's medical records in the context of Alice's name space, then the K_a prefix on Alice's policy is superfluous. The tuples from relations in other name spaces would be signed by the corresponding key and obtained from some source external to Alice's name space.

5.3.3 *Semantics.* The authors of QCM and SD3 have used a variety of formal semantics for different aspects and versions of the system. In the original presentation of QCM [Gunter and Jim 1997], the core authorization semantics are the semantics of the relational algebra. Additionally, an I/O automata model of network communication is developed to verify that certain checks during updates guarantee data consistency [Gunter et al. 1997]. In this model each QCM node maintains a set of pending queries, a set of inclusions that it accepts (initialized to the definitions known directly by the node), and a set of requests that the node has made. The automaton specifies how this state changes for each possible input or output action. A corresponding automaton models the network itself. This allows different network models, including potentially hostile models, to be studied in a straight forward way.

In later work that addresses certificate revocation, a set-theoretic model of the more recent QCM language of set comprehension is developed as a denotational semantics [Gunter and Jim 2000a]. QCM expressions are interpreted as set operations in a universe of QCM values, which include numbers, strings, keys, and finite and cofinite sets of values. An operational semantics describing the behavior of QCM evaluation is defined, and is shown to be sound with respect to the denotational semantics. It is important for the operational semantics that QCM objects are only modeled as single values or finite or cofinite sets, since this means that they can be finitely represented.

The semantics of SD3 are based on the semantics of Datalog in a fairly straightforward manner; the only complication is the interpretation of key-qualifiers on predicate names. To describe the semantics of a distributed SD3 program, first a *global Datalog program* is constructed from a given SD3 program by replacing each n-ary predicate R with an $(n+1)$-ary predicate R^g, and each atom of the form $s\$R(t_1, \ldots, t_n)$ with an atom of the form $R^g(s, t_1, \ldots, t_n)$. The semantics of an SD3 program is the minimum model of the resulting Datalog program [Jim and Suciu 2001].

While the formal meaning of QCM and SD3 programs has evolved throughout development of these systems, the authors argue that their interpretation has been essentially consistent, since relational algebra, set comprehension, and Datalog are "roughly equivalent by variations of Codd's Theorem" [Jim 2001].

5.3.4 *Implementation.* Algorithms for query processing in the QCM and SD3 systems has been defined and proven correct [Gunter and Jim 2000a; Jim 2001] with respect to a number of safety and security requirements, e.g. soundness of the algorithms with respect to the denotational meaning of programs. The distributed database approach also allows a number of standard optimization techniques to be applied, notably magic set rewriting [Jim and Suciu 2001]. Beyond this, QCM and SD3 also offer several novel implementation features.

When a QCM node is queried the result is a collection of signed tuples, possibly obtained indirectly from other nodes, forming an *extensional* response to the query. SD3 extends this by allowing a node to instead return an *intensional* response consisting of one or more rules, perhaps in terms of relations held by other nodes, that define the result of the query. In such a case the query originator could contact the other nodes if necessary to obtain the information needed to fully evaluate the query.

An example from Jim and Suciu [Jim and Suciu 2001] illustrates this distinction. Suppose that an SD3 server has the rule $R(x, y)$: - $E(x, w, z), w\$R(z, y)$ and it receives from the client the query $R(1, y)$. Suppose also that the server has the tuples $E(1, s_2, 2)$ and $E(1, s_3, 3)$ in its local table. The server could return the intensional response of

$$R(1, y) \text{ : - } s_2\$R(2, y)$$
$$R(1, y) \text{ : - } s_3\$R(3, y)$$

The client could then contact sites s_2 and s_3 to complete the query based on these rules.

Jim describes a prototype SD3 system that implements the DNSSEC protocol [Jim 2001]. In order to obtain the performance needed in DNS applications, Jim's implementation uses a number of elaborate optimization techniques. These optimizations add complexity to the implementation and increase the size of the trusted computing base. To deal with this Jim's implementation uses *certified evaluation*. The output of the SD3 query evaluator is checked by a relatively simple proof checker. If the check fails, the results of the query are considered erroneous. Since proof checking is easier than proof construction, the proof checker can be small and simple, thus reducing the size of the trusted computing base.

QCM supports distributed credential chain discovery, which the QCM authors refer to as *policy-directed certificate retrieval* [Gunter and Jim 2000b]. Such distributed queries are satisfied extensionally. For example if a QCM node a defines a relation **ratings** $=$ b.**ratings** that node would answer queries about the membership of **ratings** by querying b. If b defined its **ratings** relation in some complex way it might query other nodes as appropriate. However, node b would return signed tuples from its **ratings** relation rather than a signed policy rule.

QCM's system of distributed credential chain discovery should be contrasted with that described earlier for RT_0. In the RT_0 case distributed queries are satisfied intentionally: policy rules are passed back to the authorizing node where the entire credential chain is computed. This allows RT_0 to make direct use of credentials provided with the request without having to transmit those credentials to other nodes.

5.4 Other Trust Management Systems

In this section we review several other trust management systems more briefly, highlighting their most significant features and contributions.

5.4.1 *PolicyMaker*. Blaze et al. first introduced trust management systems per se as a subject of study [Blaze et al. 1996], by presenting the PolicyMaker system. In PolicyMaker, policies, credentials, and trust relationships between principals are implemented as arbitrary programs in a suitable safe programming language. In this context "safe" means that the interpreter for the language is restricted in terms of the I/O operations and resource consumption permitted. Such restrictions are necessary to prevent attacks against the authorization mechanism and to ensure that the authorization decision will terminate.

PolicyMaker statements, called *assertion*s, have the form: *Source* **ASSERTS** *Authority* **WHERE** *Filter*. Here *Source* and *Authority* are public keys and *Filter* is a program taking an application specific "action string" as a parameter and returning a boolean result. Policy statements have the same form except that *Source* is replaced by the keyword **POLICY** to indicate that the assertion is not a signed credential but rather locally trusted policy. More complex *Authority* structures are also possible, allowing one to express threshold policies.

The semantics of PolicyMaker is graph theoretic. Each assertion is represented as a labeled edge in directed graph G where the vertexes of G are public keys or **POLICY**. There is an edge $v_1 \rightarrow v_2$ labeled with f in G iff there is an assertion where *Source* corresponds to v_1, *Authority* corresponds to v_2 and *Filter* corresponds to f. An access request in PolicyMaker has the form: *key* **REQUESTS** *ActionString*. Authorization requires to find a path in the graph G that starts at **POLICY**, ends at the vertex corresponding to *key*, and for which $f(\textit{ActionString})$ returns true on every edge in the path.

The form and meaning of the action strings are not defined by PolicyMaker but must be agreed upon by the authorizer and the requester. For example an action string might describe a particular operation, such as read or write, on a particular file. Each assertion either allows or rejects the action according to the program contained in its filter. If the action is allowed, authority for that action is passed from the *Source* key to the *Authority* key. If there is a path in G that passes rights from **POLICY** to the requesting key, those rights are granted.

Blaze et al. formalized the PolicyMaker authorization decision and analyzed its computational complexity in later work [Blaze et al. 1998]. The general system is undecidable since the programs contained in the assertions can be written in a Turing complete language. However, Blaze et al. consider several restrictions on the system. With suitable restrictions, the authorization algorithm has polynomial complexity while retaining enough expressiveness to be useful.

5.4.2 *KeyNote*. KeyNote [Blaze et al. 1999a; 1999b] is a direct descendant of PolicyMaker. In KeyNote principals are either public keys or opaque *principal identifiers* with an application-defined meaning. In KeyNote the authorization mechanism is given a collection of assertions together with the key or identifier of the requester. The authorization mechanism returns an application defined *policy compliance value* representing the degree to which the request complies with policy.

Each KeyNote assertion specifies an authorizer and a *licensee*. As with PolicyMaker, the assertion represents a transfer of authority from the authorizer to the licensee. However, unlike PolicyMaker where the language used in the assertions is left open, KeyNote defines

a specific language. This language includes support for simple mathematical computations and string matching via regular expressions.

Although there is no formal description of KeyNote in the original presentation [Blaze et al. 1999a], KeyNote has been formally analyzed [Weeks 2001; Li and Mitchell 2003a]. Using Datalog with constraints Li and Mitchell find that KeyNote's assertion language is, in some respects, too expressive. To capture KeyNote's computational ability, a rich constraint domain is necessary. As a result, certain authorization problems are undecidable, such as determining the set of all requests that a collection of KeyNote assertions authorize.

It is instructive to consider our running example (Sect. 2.5) in KeyNote. Here the hospital database could write a policy assertion that grants all rights to Alice's medical records to Alice's key. Such an assertion might look like[3]:

```
Authorizer: POLICY
Licensees: "RSA:123abc"    # Alice's key.
Conditions: (name == "Alice")
```

A KeyNote application passes a collection of name, value pairs called *action attributes* containing information about the context of the request to the authorization mechanism. In this case, we assume the hospital database application will pass a "name" attribute identifying whose records are being accessed. It is likely that the application would pass additional attributes to KeyNote as well that provide more specific information about the request. In the KeyNote assertion above, the hospital is authorizing all requests made by Alice's key for which the name attribute is "Alice." This policy gives Alice total control over her own medical records. A more realistic policy might restrict Alice in certain ways by using more complicated conditions involving more action attributes.

Alice passes her authority over her medical records to her physician Bob by issuing (and signing) a credential such as:

```
Authorizer: "RSA:123abc" # Alice's key.
Licensees: "RSA:456def"  # Bob's key.
Conditions: (name == "Alice")
Signature: "DSA-SHA1:8912aa"
```

Again, in a more realistic situation, Alice might wish to pass only a portion of her authority to Bob.

The limitations of KeyNote (and also PolicyMaker) become apparent when Bob tries to delegate access to Alice's records to his medical team. KeyNote does not provide a language for defining and manipulating groups of principals. Thus Bob is forced to explicitly list all the keys corresponding to his medical team in the Licensees field of any assertion he writes. Without the indirection made possible by roles, policy administration in KeyNote is much more difficult than in SPKI/SDSI or RT, for example. Furthermore, when Bob consults with Carol about Alice's condition, Bob can easily write an assertion conveying his access to Alice's records to Carol. However, Carol must now write a new assertion of her own conveying that access on to Dave, her lab technician. Linked names in SDSI or linked roles in RT allow policies where this last step happens automatically; under those systems, once Carol has been granted access, she need not do anything in order for her technicians to also access Alice's records.

[3] In this example keys and signatures are abbreviated for easy presentation.

5.4.3 *REFEREE.* The REFEREE system [Chu et al. 1997] was originally considered as a trust management language by the World Wide Web Consortium's PICS (Platform for Internet Content Selection) working group [Resnick and Miller 1996] for possible use in content selection applications. The PICS effort is now subsumed by RDF (Resource Description Framework).

Like PolicyMaker, policies and credentials in REFEREE contain executable programs. However, REFEREE differs from PolicyMaker in that the execution of policies and credentials is itself put under the control of policy. In this manner REFEREE attempts to mitigate the risks associated with executing arbitrary programs as part of the authorization decision. The policy can prohibit the execution of credentials from untrustworthy sources. In addition REFEREE places signature verification and the fetching of remote credentials under policy control as well. The idea is that such actions are potentially dangerous, or at least require a certain amount of trust, and thus should be explicitly governed by the authorizer's policy. In this respect REFEREE represents some of the earliest work in automated trust negotiation, although it wasn't called that at the time.

Policy programs in REFEREE can return one of "yes", "no", or "unknown." An affirmative value implies that the policy is definitely satisfied. A negative value implies that the policy is definitely not satisfied. An "unknown" value implies that there is insufficient information to decide compliance with the policy. In this way REFEREE does not automatically deny a request when the compliance with policy is ambiguous. The application must decide how to react to an "unknown" result. Although a high security application might want to grant access only if the policy program returns an affirmative result, the web applications for which REFEREE is targeting might want a more flexible approach to ambiguous requests.

Unlike PolicyMaker, REFEREE lacks a formal specification and does not appear to have been formally analyzed in the literature. Its model of evaluation is different than PolicyMaker's in that assertions can directly invoke each other rather than executing in isolation. This allows for more complex interactions between the assertions and makes the graph-theory explanation used with PolicyMaker inapplicable.

5.4.4 *OASIS.* In OASIS [Hayton et al. 1998; Hine et al. 2000; Bacon et al. 2002; Dimmock et al. 2004] clients are classified into named roles by appropriate certificate authorities. An authorizer uses membership in a particular role as the basis for deciding access. The client collects role membership certificates from various certificate authorities ahead of time. To obtain such a certificate the client must show compliance with the authority's policy for role membership. This might require the use of previously obtained certificates or a proof of identity by way of some authentication protocol, or both. Once an appropriate role membership certificate has been obtained no further authorization computations need to be done. The authorizer simply checks the validity of the role membership certificate and grants access accordingly. Thus OASIS effectively moves the authorization computation off-line and distributes it to the various certificate authorities.

A certificates may become invalid because a certificate authority's policy might change. Alternatively the client might no longer have the necessary characteristics to be eligible for membership in a critical role. To deal with this OASIS requires that servers maintain information about every place where their certificates are used. If a certificate needs to be revoked later, the issuing server proactively contacts all servers using the certificate to inform them. In this way OASIS provides rapid response to changing conditions.

OASIS uses *appointment certificates* to provide delegation of authority and delegation of rights. A principal issues an appointment certificate to allow another principal the ability to activate a role. For example, a principal who can activate a role r can delegate the rights implied by r to another principal by issuing an appointment certificate that allows that other principal to also activate r. OASIS appointments are a generalization of normal role delegation because the issuer can appoint a subject to a role that the issuer can not activate. For example, a human resources director at a hospital can appoint a doctor without having the privileges of a doctor.

OASIS also allows arbitrary *environmental constraints* to be used in rules for enabling role activations and for maintaining role membership. The logical core of OASIS treats environmental constraints as atomic propositions. They are intended to allow the expression of policies based on time of day, local machine identity, or other similar factors. Since the environmental constraints are left unspecified, arbitrary amounts of computation could be done to evaluate them. As a consequence, the question of the tractability of access control decisions under OASIS can't be definitively answered without first making some assumptions about the nature of the environmental constraints being used.

Role activation and role maintenance rules in OASIS are definite Horn clauses and thus represent a subset of first order logic. In particular, role activation rules have the form $\Delta \vdash r$ where r is the role that may be activated and the conditions in Δ are elements in the union of roles, appointment certificates, and environmental constraints. All of the components of a rule can be parameterized. In the case of role activation such a rule means that a principal may activate role r if that principal has activated all of the roles in Δ, holds all the appointment certificates in Δ, and if all the environmental constraints in Δ are satisfied. In the case of role maintenance, such a rule means that a principal may remain in role r as long as all the conditions in Δ are satisfied. The requirements for maintaining a role activation may be less stringent than the requirements for activating the role in the first place.

OASIS does not provide a language for specifying appointments, leaving that instead to individual applications. The presumption is that a principal should have to be in a particular role to issue particular appointments but OASIS does not describe how to express that detail. For example the human resources director at a hospital would first have to activate a special role that allows her to appoint doctors. Most likely, she would be able to activate this role due to an appointment certificate she has been given by the hospital administration. However, OASIS does not provide a way to specify what appointments a principal can make in terms of the roles that principal has activated.

To encode the running example of section Sect. 2.5 in OASIS one must presuppose the existence of a policy outside of the OASIS system that defines the conditions under which appointment certificates can be issued. For example, the hospital database might specify an OASIS rule such as `records_user(X)` $\vdash$ `records(X)` allowing any entity with an appointment certificate `records_user(X)` for patient X to activate a role providing access to X's medical records. The hospital can then define a policy allowing Alice to issue appointment certificates of the form `records_user(Alice)`, although OASIS does not define what this policy would look like. Alice would use this ability to issue an appointment certificate for Bob who could then use the appointment certificate to activate the `records(Alice)` role when necessary. In order for Bob to delegate his access to Carol, Bob would need to be able to create an appointment certificate `records_user(Alice)` that Carol can use. His ability to do this depends on the hospital's policy regarding the cre-

ation of appointment certificates.

5.4.5 *PCA.* Proof Carrying Authorization (PCA) uses a higher order logic to specify both policy and credentials [Appel and Felten 1999; Bauer et al. 2002; Bauer 2003]. This logic is, in general, undecidable. However, this does not cause a problem for the authorizer because in PCA it is the requester who must construct a proof of authorization. The authorizer only needs to check this proof, something that is both decidable and tractable, to verify that the requester does have the requested access. PCA thus borrows concepts from proof carrying code [Necula 1997] where untrusted code must be accompanied by a safety proof that is checked by the consumer of that code.

This approach seems to put a significant burden on requesters. However, each requester normally only needs to work with a subset of the full logic. For any particular application, an application-specific logic can be defined where the rules of inference in that logic are lemmas in the general higher order logic. The requester can construct a proof using this limited logic but the authorizer does not need to be aware of the particular application-specific logic being used. The requester provides proofs of the necessary application-specific lemmas as part of the proof of authorization.

Many application-specific logics are possible. Other trust management systems such as SPKI can be encoded as an application-specific logic for use with PCA. Thus a PCA authorizer is able to work with requesters using a variety of trust management methods in a uniform way. In this respect PCA is a generalization of the other systems reviewed here.

Principals in PCA are modeled as sets of formulae the principal regards as true and thus can be represented as a higher order predicate taking a formula as an argument. P is a principal if both $\forall F.F \supset P(F)$ and $\forall F_1 \forall F_2.(P(F_1) \wedge P(F_1 \supset F_2)) \supset P(F_2)$. In other words, P is a principal if P admits true statements and any statement that is implied by other statements P admits.

The logic contains a primitive `signed` connective used to represent digitally signed statements, and constants used to represent keys. As an example, consider the following three formulae.

(1) K_{ca} signed $(\forall F.(K_a \text{ signed } F) \supset (\textit{Alice}(F)))$

(2) $\forall F.(K_{\text{ca}} \text{ signed } F) \supset \textit{CA}(F)$

(3) $\forall k. \forall p. \textit{CA}((\forall S.(k \text{ signed } S) \supset p(S))) \supset \forall S.(k \text{ signed } S) \supset p(S)$

The first formula describes the binding of the key K_a to the principal *Alice* made by an authority with key K_{ca}. It asserts that any formula signed by K_a is admitted by the principal *Alice*. This formula plays the role of an identity certificate. The second formula is part of the authorizer's policy. It asserts that any formula signed by K_{ca} can be attributed to the *CA* principal. In other words, K_{ca} is the correct key. The final formula is how the authorizer delegates authority to *CA*. It asserts that if the certificate authority asserts that k is principal p's key, then the authorizer will take any statement signed by k as a statement admitted by p.

The beauty of the PCA system is that formula such as the ones above are not built into the system but instead are constructed to suit the needs of a particular application. For this reason most of the features we describe for trust management systems are supported by PCA indirectly.

The inference rules of PCA those of higher order logic along with a few additional rules that talk about keys and digital signatures. These additional rules are shown in Fig. 6. The

$$\text{NAME\_I}\ \frac{F}{\mathcal{N}(k)(F)} \qquad \text{NAME\_IMP\_E}\ \frac{\mathcal{N}(k)(F) \qquad \mathcal{N}(k)(F \supset G)}{\mathcal{N}(k)(G)} \qquad \text{SIGNED}\ \frac{\text{digital\_signature}(s, k, F)}{\mathcal{N}(k)(F)}$$

Fig. 6. Some Inference Rules of PCA Logic

function $\mathcal{N}$ takes a string, for example a public key, and returns the principal corresponding to that key. The NAME_I and NAME_IMP_E rules embody the definition of a principal mentioned above. The SIGNED rule says that a formula signed by key k is a statement admitted by principal $\mathcal{N}(k)$.

PCA uses an interesting mechanism to handle certificate revocation [Bauer 2003], that is an example of how this feature can be handled while preserving monotonicity; their approach is inspired by previous work [Li and Feigenbaum 2002] proposed for use in the RT framework [Li et al. 2002].

In particular, a PCA implementation only treats a certificate as valid if there is an appropriate certificate revocation list available. The presence of a revocation list does not remove a previously valid certificate, rather it enables a certificate that was previously invalid to be used. The relevant inference rule is as follows, though the PCA authors note that this rule is derivable as a theorem:

$$\text{CERT-E}\ \frac{\text{cert}(A, F, N) \qquad A\ \text{signed}(\text{revlist}(T_1, T_2, L)) \qquad \text{localtime} < T_2 \qquad N \notin L}{A\ \text{says}\ F}$$

This rule states that principal A says F provided there is a certificate asserting it and a revocation list signed by A that is currently valid and which does not include the certificate's serial number. If the revocation list is not available, A says F can not be deduced from the certificate alone.

5.4.6 *TPL.* In Trust Policy Language (TPL) [Herzberg et al. 2000] the policy language and the certificate language are distinct. Certificates bind attributes to public keys and can be translated from other certificate formats, such as X.509v3. The policy language allows authorizers to define rules, based on certificate attribute values, by which an entity, represented by a public key, can enter a role. As with RT but unlike SPKI, the system does not directly express authorizations. Instead only role memberships are computed. The permissions granted to an entity depend on the resulting role memberships and are defined externally.

TPL uses an XML syntax for its policy language. The example in Fig. 7 [Herzberg et al. 2000] illustrates a policy statement that defines the members of a group (or role) named "Hospitals." In such statements multiple rules are allowed; if any of the rules are satisfied then the policy is satisfied. A rule contains one or more INCLUSION elements, each of which represents a certificate or, if the REPEAT attribute is present, multiple certificates. In the example a subject is a member of the Hospitals group if that subject has been recommended by two other hospitals. The FUNCTION element describes additional conditions on the various certificates in the rule. In the example, the recommendation certificates must have a "Level" field with a value greater than one.

TPL policies can be compiled to Prolog, using appropriate functions in Prolog to capture the full expressiveness of the policy language. However the implementation does not use

```
<GROUP NAME="Hospitals">
   <RULE>
      <INCLUSION
         ID="reco" TYPE="Recommendation" FROM="Hospitals" REPEAT="2"/>
      <FUNCTION>
         <GT>
            <FIELD ID="reco" NAME="Level"/>
            <CONST>1</CONST>
         </GT>
      </FUNCTION>
   </RULE>
</GROUP>
```

Fig. 7. Example TPL Policy Statement

Prolog directly but instead provides its own policy engine. Unlike an ordinary Prolog theorem prover, this engine is capable of fetching remote certificates as needed and thus provides a form of distributed chain discovery. For example, suppose X issues a certificate about some subject Y. The certificate contains an *issuerCertRepository* field where the policy engine can find more information about X (for example, the groups of which X is a member) and a *subjectCertRepository* field where the policy engine can find more information about Y (for example, the groups defined by Y).

The general form of TPL allows for credential negation and is therefore nonmonotonic. Since the requester cannot be expected to willingly provide information that would deny access, such certificates are fetched from repositories defined by the authorizer, instead of by the issuer or subject.

5.4.7 *Binder.* Like SD3, Binder [DeTreville 2002a; 2002b] uses an extended version of Datalog as its foundation. The authorizer writes Datalog rules and facts to describe the local policy. These rules and facts exist in a *context* that is associated with a public/private key pair. Rules and facts can be exported from a context by signing them. Thus signed Datalog statements form the credentials in the Binder system. An importing context quotes the credentials using *says* in a way similar to other ABLP inspired logics. Special rules in the policy must be provided to relate quoted predicates to local predicates.

The following example [DeTreville 2002a] allows all members of `bigco` to read `resource_r`. The authorizer's local policy is

$$\textit{can}(X, \texttt{read}, \texttt{resource\_r}) \leftarrow \textit{employee}(X, \texttt{bigco}).$$
$$\textit{employee}(X, \texttt{bigco}) \leftarrow K_b \textbf{ says } \textit{employee}(X, \texttt{bigco}).$$

This policy grants read access to all objects X for which the local predicate *employee*-$(X, \texttt{bigco})$ is true. The policy then connects the predicate *employee* in the context controlled by key K_b to the local predicate with the same name. When a signed credential K_b **says** *employee*$(\texttt{john}, \texttt{bigco})$ is presented to the authorizer, the authorizer can then compute that *can*$(\texttt{john}, \texttt{read}, \texttt{resource\_r})$.

In effect Binder distributes a large Datalog program over many contexts and allows each context to explicitly decide which statements from other contexts it will accept. Like SD3 this gives Binder the full expressivity of Datalog.

Binder makes no attempt to treat negative information and revocation is handled in T_C (Fig. 1) by controlling certificate lifetimes or by requiring the use of on-line revocation

checks. In addition an authorizer's policy can be, at least potentially, signed and published. Thus a Binder based system could require clients to find the necessary proofs with the authorizer simply checking the results as is done with PCA. This can help off-load work from the authorizer but the technique is not specific to Binder.

5.4.8 *Cassandra.* Cassandra [Becker and Sewell 2004a; 2004b; Becker 2005] uses a semantics based on Datalog with constraints but allows the constraint domains to be selected independently of the base system. This allows an application to tune Cassandra by trading off expressiveness in the policy language for computational efficiency without having to modify the core implementation.

In a Cassandra system, each host runs a Cassandra service. In addition to requesting access to resources, clients of the service can activate or deactivate roles in that service as well as request credentials for use with Cassandra services on other nodes. Each Cassandra service runs an authorization mechanism that consults local policy and that also makes remote queries to other Cassandra services to obtain relevant policy information. Information about where remote credentials can be found is encoded in the rules themselves; credential chain discovery is not completely automatic.

Cassandra is role based and allows roles and actions (permissions) to be parameterized. The base system uses only a few predicates including: *permits*($\texttt{entity}, \texttt{permission}$), *canActivate*($\texttt{entity}, \texttt{role}$), and *hasActivated*($\texttt{entity}, \texttt{role}$). Users are able to introduce application specific predicates as well. Whenever a role is activated in a particular Cassandra service, that service adds an appropriate *hasActivated* fact to its policy. Thus the set of policy rules available to the authorization mechanism varies as roles are activated and deactivated.

The predicates in a rule can be annotated with information about the location where specific certificates can be obtained. These annotations can be variables that are instantiated during the evaluation process. Using side-effect free functions in an integer order constraint domain, Cassandra can directly express rules regarding credential validity. For example:

$$\begin{aligned} &\textit{canActivate}(X, \texttt{Doc}()) \leftarrow \\ &\quad \textit{canActivate}(X, \texttt{CertDoc}(T)), \texttt{CurTime}() - \texttt{Years}(1) \leq T \leq \texttt{CurTime}() \end{aligned}$$

This rule says that X can activate the doctor role provided that X was certified at time T (X can activate the $\texttt{CertDoc}$ role for T), and that T is not more than one year old. In this case the authorizer defines a validity period on X's certification and won't accept certifications that are too old. Notice that in most systems the lifespan of a certificate is set by the issuer. However, since the authorizer is the principal assuming the risk of using an invalid certificate it makes sense in many applications for the authorizer to define the acceptable lifespans [Rivest 1998].

Cassandra uses an authorization procedure that is a variation of Toman's memoing algorithm for Datalog with constraints [Toman 1997]. This approach is based on SLG resolution and is goal oriented (top down) while avoiding the non-termination problems that might arise using a traditional SLD style evaluation.

6. OTHER COMPONENTS: TRUST NEGOTATION

In this survey we have focused specifically on authorization in trust management. However, modern trust management systems include other major components, to address problems

other than the semantics of authorization. While space considerations prevent a complete review of issues and approaches, in this section we provide a brief overview of *trust negotiation*, which is a topic of considerable interest in modern trust management research. Along with providing a more complete view of trust management, the purpose of this section is to provide a better practical context for the topics covered in this survey, and a better picture of current research directions.

When considering the semantics of authorization, it is simplest to assume that requesters' credentials are publicly available, so that authorizers have full access to them as well as their own policy. We call this the *basic model*, and it is typically assumed in most of the systems we survey here for the initial development of an authorization semantics. In the basic model authorizers do not directly disclose their policy to any requester while requesters are assumed to disclose their credentials freely. Requesters either send their credentials with each request or, in some cases, make their credentials available on public servers where authorizers can locate them using some form of credential chain discovery.

Furthermore, the result of the authorization decision in the basic model is a simple boolean value specifying if access is allowed or not. If access is allowed the requester does not know which credentials were actually necessary to gain that access. If access is denied the requester does not know why the denial occurred and has no way of knowing how to obtain missing credentials.

However, in practice the basic model is not always sufficient. For example, rather than expending overhead on distributed chain discovery, an authorizer may associate additional information with credentials that associates them with particular resources, so requesters can supply a subset of their credentials in an informed manner. But a deeper issue is that a requester may regard some of her credentials as sensitive and have a complex access policy for them. The requester may require an authorizer comply with that policy before she is willing to disclose those credentials. In addition the authorizer may wish to control access to his resource access policy, allowing some or all of that policy to be disclosed to suitable requesters. In this situation, the requester and authorizer can engage in a process called trust negotiation during which credentials and policy statements are shared incrementally between the authorizer and requester as the parties gain increasing trust in each other.

An example in [Seamons et al. 2002] illustrates the concepts. Alice, a university student, orders her textbooks from an online bookstore. She requests a student discount not knowing what credentials the bookstore will require. In response the bookstore asks to see her digital student ID and her digital credit card. Alice is willing to disclose her ID but will only disclose her credit card to web sites that have been certified by the Better Business Bureau; accordingly she requests this credential from the bookstore site. Once the bookstore discloses its Better Business Bureau credential, Alice's access policy on her credit card is satisfied and she discloses her credentials as well.

Many trust negotiation systems have been described in the literature [Winslett et al. 2002; Gavriloaie et al. 2004; Bonatti and Olmedilla 2005b]. Some, such as PeerTrust [Gavriloaie et al. 2004], are extensions of other trust management systems, including systems we have surveyed here. For example, PeerTrust extends SD3 with trust negotiation, a trust negotiation framework has been developed for the RT system [Winsborough and Li 2002]. In other cases, such as with Protune [Bonatti and Olmedilla 2005b], support for trust negotiation and trust management were designed together from scratch. In either case trust negotiation systems build on trust management concepts and thus have many overlapping concerns.

In a survey of trust negotiation systems a list of requirements on trust negotiation policy languages is given in [Seamons et al. 2002]. Because trust negotiation systems include trust management functionality that we survey here, many of the these policy language requirements overlap with or are embedded in our list of trust management features. For example, Seamons et al. note the importance of using languages with well defined semantics, and discuss features such as monotonicity, credential chains, delegation depth, and local name spaces. However, Seamons et al. also require trust negotiation policy languages to have the power to express access control information on the policies themselves. This is the essence of trust negotiation.

In addition [Seamons et al. 2002] gives requirements on compliance checkers. In a trust negotiation context compliance checkers can no longer return a simple yes/no result. If the request for access fails, the checker must provide information about what additional credentials are needed to gain that access. The authorizer can use this information to request those credentials from the requester. Furthermore the requester uses a compliance checker to control access to her credentials, and in some cases to locally process policy information provided by the authorizer to determine which of her credentials might be relevant to a particular request.

Some of the earliest work on trust negotiation focused on how the requester could select precisely the credentials necessary for the desired access and thus avoid sending sensitive credentials unnecessarily. In [Seamons et al. 1997] *credential acceptance policies* written in a restricted form of Prolog are downloaded from the authorizer by the requester and then executed with the requester's database of credentials as part of the collection of facts available to the Prolog program. The result is a list of required credentials that the requester must send to the authorizer to gain access to the protected resource. The authorizer executes essentially the same program to check the access request.

Later work generalizes this process by allowing the requester to assign *credential access* or *credential disclosure* policies to control the conditions under which those credentials can be revealed [Winsborough et al. 2000; Yu et al. 2000]. In addition authorizers might want to control the disclosure of their access policies as well [Seamons et al. 2001]. When a negotiating party sends all the credentials or policy statements for which the access policy has been met, that party is said to follow an *eager* strategy. On the other hand if the negotiating party only sends a more narrowly focused collection of credentials in response to specific requests from the negotiating partner, that party is said to follow a *parsimonious* strategy. In this later case, however, a negotiating party can inadvertantly reveal sensitive information about her credentials indirectly by way of her reactions to specific requests [Winsborough et al. 2000; Winsborough and Li 2002; 2004]. Fundamentally this problem arises because credential access policies are only associated with credentials a negotiating party actually has. To address this issue *ack policies* can be created that cause a negotiating party to enter into a negotiation about attributes she considers sensitive even if she lacks any specific credentials about those attributes [Winsborough and Li 2002; 2004].

The PeerTrust system is an example of a fully developed trust negotation policy system. PeerTrust extends SD3 to provide trust negotiation [Gavriloaie et al. 2004]. The Horn clauses used in SD3 are enhanced to allow guards. Essentially the body of a clause is broken into sections where the successful evaluation of one section is required before the evaluation of the next section is allowed to begin. An example in [Gavriloaie et al. 2004] shows a rule Alice might use to specify that she will only reveal her signed credential that she is a California state police officer to web sites that are members of the Better Business

Bureau:

policeOfficer(alice) @ CSP ←
member(Requester) @ BBB @ Requester | signedBy[CSP].

In this rule the "@" is used to indicate a non-local predicate. The ability to evaluate non-local predicates nests so that, for example, "member(Requester) @ BBB @ Requester" means that the requester must show that he is a member of the Better Business Bureau. In this context the requester is a web server asking Alice if she is a police officer. This predicate must be validated before the evaluation of the rule can proceed beyond the "|" and Alice can send the necessary signed certificate to the web server.

For another example, PROTUNE is a particularly rich trust negotiation system providing support for authentication, policy rules with actions and side effects (*provisional* rules), as well as hierarchical services and credentials [Bonatti and Samarati 2000; Bonatti and Olmedilla 2005b]. Metapolicies are used to arbitrate trust negotiation; as credentials are gathered a negotiating party's metapolicies activate new policy rules as appropriate. PROTUNE has also been extended to provide support for advanced queries allowing users to ask high level questions about authorization decisions such as, for example, *why?*, *what if?*, and *how to?* queries [Bonatti et al. 2006].

While PROTUNE provides extensive support for trust negotiation, at its core it also contains support for many of the authorization features we discuss in this work. PROTUNE libraries can be created that simulate the semantics of other trust management systems. For example, PROTUNE can encode the four credential forms of RT_0 [Bonatti and Olmedilla 2005a], thus providing all the capabilities of that system such as local name spaces, role-based access control, and delegation of rights. In addition PROTUNE's features can be used to encode a public key infrastructure [Bonatti and Samarati 2000] and distributed chain discovery [Bonatti and Olmedilla 2005a]. The last column in Table II compares PROTUNE with the other systems we review in terms of its authorization semantics.

7. CONCLUSION

Trust management technology responds to the security demands of modern distributed systems– or in the words of previous authors [Blaze et al. 1999b], "the trust management approach to distributed system security was developed as an answer to the inadequacy of traditional authorization mechanisms". Compared to other simpler systems such as identity or role-based access control, trust management systems provide modern distributed applications programmers a more effective and scalable means of specifying and enforcing authorization policies. At the heart of any trust management system is a language for expressing policy and access rights, comprising a mixture of features that address the character and requirements of modern distributed security. We have reviewed a number of language features, and summarized which systems possess which features (see especially Tables I and II). Overall, we conclude that a subset of them are fundamental to any trust management system:

Linked local namespaces. No single, monolithic namespace exists on the Internet. Instead, distinct security domains define their own namespaces. Any trust management system language must provide some means to refer to non-local namespaces, within the local namespace.

Roles. Role membership is a fundamental abstraction in trust management systems. The specification of authorization via roles allows policies to be defined independently of iden-

tities, so that addressing the needs of unknown future users does not require policy to be rewritten.

Delegation of authority. No single, monolithic policy authority exists on the Internet. Rather, distinct security domains define their own local policies, and security domains commonly delegate authority over local policy to trusted non-local authorities; any trust management system language must provide some way to express this.

Delegation of rights. In many cases, the users of a system desire to delegate their access rights to other entities, to act on their behalf. trust management system languages should address this need, including important nuances such as whether certain rights should be specified as undelegatable by local authorities.

Certificate revocation. Access rights should never be permanently granted, rather authorities should be able to revoke them or set finite lifetimes for their use. While certificate revocation appears to be an essentially nonmonotonic feature, this has been disproven, as the authors of both PCA [Bauer 2003] and RT [Li and Feigenbaum 2002] have developed monotonic inference rules for incorporating revocation in authorization. Some systems such as SPKI/SDSI offer a simpler implementation-based solution [Ellison et al. 1999], where revocation is not featured in the authorization language but processed during the parsing of certificates into credentials.

At a higher level, we argue that rigorous formal foundations are a necessary design feature of trust management systems, since they allow rigorous guarantees of security. We have shown that graph theories, logics, and database formalisms are the most common formalisms for trust management system design, though logic stands out as the most popular. Indeed, previous authors have argued that monotonic programming logics are uniquely well-suited as trust management languages [Li and Mitchell 2003a], and in general the rich semantic domains available in logic provide a great deal of flexibility and scalability in trust management system applications [Polakow and Skalka 2006]. A variety of trust management system foundations use domain-specific logical constructs originally developed for authentication settings [Burrows et al. 1990], witnessing the evolution of authorization systems from earlier access control systems based mainly on authentication.

Many of the systems surveyed in this paper exist primarily in theory, and have not yet been deployed. As trust management systems become more commonplace, and theory develops into practice, we believe a major challenge will be whole-system assurances. As we have discussed, trust management systems comprise more than just a semantics for their core authorization language, including a collection of features for storing, collecting, and processing certificates. Significantly, some trust management system features are sometimes implemented in these other components, such as when certificate revocation or expiration is realized during certificate parsing. And of course, correctness of an entire system depends on correctness of all of its components, as well as their interaction. Thus, formalisms for assurances of correctness of *systems* must transcend the core authorization semantics, and address the myriad components of trust management systems. In many ways QCM and SD3 set the standard in this regard [Gunter and Jim 1997; Jim 2001], by using database theory and technology as a uniform setting for implementing certificate storage and retrieval, defining the semantics of authorization, and formally modeling systems. As trust management systems mature into vital components of Internet communications and commerce, a holistic formal view that takes into account the variety of trust

management system components will be essential to coherence and reliability of these systems.

REFERENCES

ABADI, M. 1998. On SDSI's linked local name spaces. *Journal of Computer Security 6*, 1–2, 3–21.

ABADI, M. 2003. Logic in access control. In *Proceedings of the 18th IEEE Symposium on Logic in Computer Sciennce*.

ABADI, M., BURROWS, M., LAMPSON, B., AND PLOTKIN, G. 1993. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems 15*, 4 (September), 706–734.

AJMANI, S., CLARKE, D. E., MOH, C.-H., AND RICHMAN, S. 2002. ConChord: Cooperative SDSI certificate storage and name resolution. In *International Workshop on Peer-to-Peer Systems*.

APPEL, A. W. AND FELTEN, E. W. 1999. Proof-carrying authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*. ACM Press, New York, NY, USA, 52–62.

BACON, J., MOODY, K., AND YAO, W. 2002. A model of OASIS role-based access control and its support for active security. *ACM Transactions on Information and System Security 5*, 4 (November), 492–540.

BAUER, L. 2003. Access control for the web via proof-carrying authorization. Ph.D. thesis, Princeton University.

BAUER, L., SCHNEIDER, M. A., AND FELTEN, E. W. 2002. A general and flexible access-control system for the web. In *Proceedings of the 11th USENIX Security Symposium*. 93–108.

BECKER, M. Y. 2005. Cassandra: Flexible trust management and its application to electronic health records. Tech. Rep. 648, University of Cambridge. October.

BECKER, M. Y. AND SEWELL, P. 2004a. Cassandra: Distributed access control policies with tunable expressiveness. In *Proceedings of the 5th IEEE International Workshop on Policies for Distributed Systems and Networks*.

BECKER, M. Y. AND SEWELL, P. 2004b. Cassandra: Flexible trust management, applied to electronic health records. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*.

BERTINO, E., CATANIA, B., FERRARI, E., AND PERLASCA, P. 2003. A logical framework for reasoning about access control models. *ACM Transactions on Information and System Security 6*, 1 (February), 71–127.

BLAZE, M., FEIGENBAUM, J., IOANNIDIS, J., AND KEROMYTIS, A. D. 1999a. *RFC-2704: The KeyNote Trust-Management System Version 2*. Internet Engineering Task Force.

BLAZE, M., FEIGENBAUM, J., IOANNIDIS, J., AND KEROMYTIS, A. D. 1999b. The role of trust management in distributed systems security. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*. Springer-Verlag, London, UK, 185–210.

BLAZE, M., FEIGENBAUM, J., AND LACY, J. 1996. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 164–173.

BLAZE, M., FEIGENBAUM, J., AND STRAUSS, M. 1998. Compliance checking in the policymaker trust management system. In *Proceedings of the Second International Conference on Financial Cryptography*. Springer-Verlag, 254–274.

BLAZE, M., IOANNIDIS, J., AND KEROMYTIS, A. D. 2002. Trust management for IPsec. *ACM Transactions on Information and System Security 5*, 2 (May), 95–118.

BLAZE, M., IOANNIDIS, J., AND KEROMYTIS, A. D. 2003. Experience with the keynote trust management system: Applications and future directions. In *Proceedings of the First International Conference on Trust Management*. Springer-Verlag, Keraklion, Crete, Greece, 284–300.

BONATTI, P. AND OLMEDILLA, D. 2005a. Policy language specification. REWERSE Deliverable I2-D2, February 2005. `http://rewerse.net/deliverables.html`.

BONATTI, P. AND SAMARATI, P. 2000. Regulating service access and information release on the web. In *Proceedings of the 7th ACM conference on computer and communications security*. ACM Press, 134–143.

BONATTI, P. AND SAMARATI, P. 2003. Logics for authorizations and security. In *Logics for Emerging Applications of Databases*, J. Chomicki, R. van der Meyden, and G. Saake, Eds. Springer-Verlag.

BONATTI, P. A. AND OLMEDILLA, D. 2005b. Driving and monitoring provisional trust negotiation with metapolicies. In *IEEE 6th International Workshop on Policies for Distributed Systems and Networks*. Stockholm, Sweden.

BONATTI, P. A., OLMEDILLA, D., AND PEER, J. 2006. Advanced policy queries. In *Proceedings of the 17th European Conference on Artificial Intelligence*. IOS Press, 200–204.

BURROWS, M., ABADI, M., AND NEEDHAM, R. M. 1990. A logic of authentication. *ACM Transactions on Computer Systems 8*, 1 (February), 18–36.

CHU, Y.-H., FEIGENBAUM, J., LAMACCHIA, B., RESNICK, P., AND STRAUSS, M. 1997. REFEREE: Trust management for web applications. *World Wide Web Journal 2*, 3 (Summer), 127–139.

CLARKE, D., ELIEN, J.-E., ELLISON, C., FREDETTE, M., MORCOS, A., AND RIVEST, R. L. 2001. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security 9*, 4, 285–322.

DETREVILLE, J. 2002a. Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. IEEE Computer Society.

DETREVILLE, J. 2002b. Making certificates programmable. In *Proceedings of the First Annual PKI Workshop*. Hanover, NH, USA.

DIMMOCK, N., BELOKOSZTOLSZKI, A., EYERS, D., BACON, J., AND MOODY, K. 2004. Using trust and risk in role-based access control policies. In *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies*. ACM Press, New York, NY, USA, 156–162.

EITER, T., GOTTLOB, G., AND MANNILA, H. 1997. Disjunctive datalog. *ACM Trans. Database Syst. 22*, 3, 364–418.

ELLISON, C., FRANTZ, B., LAMPSON, B., RIVEST, R., THOMAS, B., AND YLONEN, T. 1999. *RFC-2693: SPKI Certificate Theory*. Internet Engineering Task Force.

FERRAIOLO, D. AND KUHN, R. 1992. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*. 554–563.

GAVRILOAIE, R., NEJDL, W., OLMEDILLA, D., SEAMONS, K. E., AND WINSLETT, M. 2004. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *Proceedings of the 1st European Semantic Web Symposium*. Lecture Notes in Computer Science, vol. 3053. Springer, Heraklion, Crete, Greece, 342–356.

GUNTER, C. A. AND JIM, T. 1997. Design of an application-level security infrastructure. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*.

GUNTER, C. A. AND JIM, T. 2000a. Generalized certificate revocation. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 316–329.

GUNTER, C. A. AND JIM, T. 2000b. Policy-directed certificate retrieval. *Software: Practice & Experience 30*, 15, 1609–1640.

GUNTER, C. A., JIM, T., AND WANG, B.-Y. 1997. Authenticated data distribution using query certificate managers. unpublished extended abstract.

HALPERN, J. AND VAN DER MEYDEN, R. 1999. A logic for SDSI's linked local name spaces. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*. 111–122.

HALPERN, J. Y. AND VAN DER MEYDEN, R. 2001. A logical reconstruction of SPKI. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*. 59–70.

HAYTON, R. J., BACON, J. M., AND MOODY, K. 1998. OASIS: Access control in an open distributed environment. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 3–14.

HERZBERG, A., MASS, Y., MICHAELI, J., NAOR, D., AND RAVID, Y. 2000. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the IEEE Symposium on Security and Privacy*.

HINE, J. A., YAO, W., BACON, J., AND MOODY, K. 2000. An architecture for distributed OASIS services. In *Middleware 2000: IFIP/ACM International Conference on Distributed Systems Platforms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 104–120.

HOWELL, J. 2000. Naming and sharing resrouces across administrative boundaries. Ph.D. thesis, Dartmouth College.

HOWELL, J. AND KOTZ, D. 2000. A formal semantics for SPKI. Tech. Rep. 2000-363, Dartmouth College.

International Telecommunications Union 2000. *Information Technology - Open Systems Interconnection - The Directory: Public Key and Attribute Certificate Frameworks*. International Telecommunications Union.

International Telecommunications Union 2001. *Information Technology - Open Systems Interconnection - The Directory: Overview of Concepts, Models, and Services*. International Telecommunications Union.

JAFFAR, J. AND MAHER, M. J. 1994. Constraint logic programming: A survey. *Journal of Logic Programming 19/20*, 503–581.

JHA, S. AND REPS, T. 2002. Analysis of SPKI/SDSI certificates using model checking. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*. IEEE Computer Society, Washington, DC, USA, 129.

JIM, T. 2001. SD3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*. IEEE Computer Soceity.

JIM, T. AND SUCIU, D. 2001. Dynamically distributed query evaluation. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM Press, New York, NY, USA, 28–39.

LI, N. 2000. Local names in SPKI/SDSI. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, Cambridge, UK, 2–15.

LI, N. AND FEIGENBAUM, J. 2002. Nonmonotonicity, user interfaces, and risk assessment in certificate revocation. In *Proceedings of the 5th International Conference on Financial Cryptography*. Springer-Verlag, London, UK, 166–177.

LI, N., GROSOF, B. N., AND FEIGENBAUM, J. 2003. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security 6*, 1 (February), 128–171.

LI, N. AND MITCHELL, C. 2006. Understanding spki/sdsi using first-order logic. *International Journal of Information Security 5*, 1, 48–64.

LI, N. AND MITCHELL, J. C. 2003a. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages*.

LI, N. AND MITCHELL, J. C. 2003b. RT: A role-based trust-management framework. In *Proceedings of the Third DARPA Information Survivability Conference and Exposition*. IEEE Computer Society Press, 201–212.

LI, N., MITCHELL, J. C., AND WINSBOROUGH, W. H. 2002. Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 114–130.

LI, N., MITCHELL, J. C., AND WINSBOROUGH, W. H. 2005. Beyond proof-of-compliance: Security analysis in trust management. *Journal of the ACM 52*, 3 (May), 474–514.

LI, N., WINSBOROUGH, W. H., AND MITCHELL, J. C. 2003. Distributed chain discovery in trust management. *Journal of Computer Security 11*, 1 (Feb), 35–86.

LIU, Y. D. AND SMITH, S. 2002. A component security infrastructure. In *Proceedings of the 2002 Foundations of Computer Security Workshop*.

MCDANIEL, P. AND RUBIN, A. D. 2001. A response to "can we eliminate certificate revocation lists?". In *Proceedings of the 4th International Conference on Financial Cryptography*. Springer-Verlag, London, UK, 245–258.

NECULA, G. C. 1997. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of programming languages*. ACM Press, New York, NY, USA, 106–119.

NIKANDER, P. AND VILJANEN, L. 1998. Storing and retrieving internet certificates. In *Proceedings of the Third Nordic Workshop on Secure IT Systems*.

OASIS. 2006a. OASIS eXtensible Access Control Markup Language Technical Committee at `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`.

OASIS. 2006b. OASIS Security Services Technical Committee at `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security`.

OASIS. 2006c. OASIS Web Services Security Technical Committee at `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss`.

OFFICE OF TECHNOLOGY ASSESSMENT. 1993. Protecting privacy in computerized medical information, OTA-TCT-576. U.S. Government Printing Office.

POLAKOW, J. AND SKALKA, C. 2006. Specifying distributed trust management in LolliMon. In *Proceedings of the ACM Workshop on Programming Languages and Analysis for Security*.

RESNICK, P. AND MILLER, J. 1996. PICS: Internet access controls without censorship. *Communications of the ACM 39*, 10 (October), 87–93.

RIVEST, R. L. 1998. Can we eliminate certificate revocation lists? In *Proceedings of the Second International Conference on Financial Cryptography*. Springer-Verlag, London, UK, 178–183.

RIVEST, R. L. AND LAMPSON, B. 1996a. SDSI — A Simple Distributed Security Infrastructure. Version 1.0, at `http://theory.lcs.mit.edu/~rivest/sdsi10.html`, September 15, 1996.

RIVEST, R. L. AND LAMPSON, B. 1996b. SDSI — A Simple Distributed Security Infrastructure. Version 1.1, at `http://theory.lcs.mit.edu/~rivest/sdsi11.html`, October 2, 1996.

SANDHU, R. S., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. 1996. Role-based access control models. *Computer 29*, 2, 38–47.

SEAMONS, K., WINSBOROUGH, W., AND WINSLETT, M. 1997. Internet credential acceptance policies. In *Proceedings of the Workshop on Logic Programming for Internet Applications*. Leuven, Belgium.

SEAMONS, K., WINSLETT, M., AND YU, T. 2001. Limiting the disclosure of access control policies during automated trust negotiation.

SEAMONS, K., WINSLETT, M., YU, T., SMITH, B., CHILD, E., JACOBSON, J., MILLS, H., AND YU, L. 2002. Requirements for policy languages for trust negotiation. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks*. IEEE Computer Society, Washington, DC, USA, 68.

SIMON, R. T. AND ZURKO, M. E. 1997. Separation of duty in role-based environments. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 183–194.

SKALKA, C., WANG, X. S., AND CHAPIN, P. 2007. Risk management for distributed authorization. *Journal of Computer Security*. To appear.

STUBBLEBINE, S. 1995. Recent-secure authentication: Enforcing revocation in distributed systems. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 224–235.

STUBBLEBINE, S. G. AND WRIGHT, R. N. 1996. An authentication logic supporting synchronization, revocation, and recency. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*. ACM Press, New York, NY, USA, 95–105.

TOMAN, D. 1997. Memoing evaluation for constraint extensions of datalog. *Constraints 2*, 337–359.

WEEKS, S. 2001. Understanding trust management systems. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*. IEEE Computer Society, Washington, DC, USA, 94.

WINSBOROUGH, W. H. AND LI, N. 2002. Towards practical automated trust negotiation. In *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*. IEEE Press.

WINSBOROUGH, W. H. AND LI, N. 2004. Safety in automated trust negotiation. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*. IEEE Computer Society, Los Alamitos, CA, USA, 147.

WINSBOROUGH, W. H., SEAMONS, K. E., AND JONES, V. E. 2000. Automated trust negotiation. In *Procedings of the DARPA Information Survivability Conference and Exposition. Volume 1*. IEEE Computer Society, 88–102.

WINSLETT, M., CHING, N., JONES, V., AND SLEPCHIN, I. 1997. Assuring security and privacy for digital library transactions on the web: Client and server security policies. In *Proceedings of the IEEE International Forum on Research and Technology Advances in Digital Libraries*. IEEE Computer Society, Washington, DC, USA, 140–151.

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

WOBBER, E., ABADI, M., BURROWS, M., AND LAMPSON, B. 1993. Authentication in the taos operating system. *SIGOPS Operating Systems Review. 27*, 5, 256–269.

WOO, T. Y. C. AND LAM, S. S. 1993. Authorizations in distributed systems: A new approach. *Journal of Computer Security 2*, 2-3, 107–136.

XSB INC. 2006. XSB home page. `http://xsb.sourceforge.net`.

YU, T., MA, X., AND WINSLETT, M. 2000. PRUNES: An efficient and complete strategy for automated trust negotiation over the internet. In *Proceedings of the 7th ACM conference on Computer and communications security*. ACM Press, New York, NY, USA, 210–219.

YU, T., WINSLETT, M., AND SEAMONS, K. E. 2001. Interoperable strategies in automated trust negotiation. In *Proceedings of the 8th ACM conference on Computer and Communications Security*. ACM Press, New York, NY, USA, 146–155.