Effective Trust Management Through a Hybrid Logical and Relational Approach

Adam J. Lee[†]
adamlee@cs.pitt.edu

Ting Yu[‡]
yu@csc.ncsu.edu

Yann Le Gall[†]
ygl2@pitt.edu

[†]Department of Computer Science, University of Pittsburgh
[‡]Department of Computer Science, North Carolina State University

ABSTRACT

Despite a plethora of recent research regarding trust management approaches to authorization, relatively little attention has been given to exactly how these technologies can be effectively deployed. In this paper, we investigate one way in which well-established logical trust management systems described in the literature can be deployed within enterprise environments. Specifically, we develop a framework within which logical trust management policies can be managed using a relational DBMS. We describe a correct and complete procedure for compiling CTM credentials into dynamic views within a database, and show how the resulting system can be used to perform role membership checks or to enumerate the members of a given role. We then propose a hybrid algorithm that leverages the logical ruleset and the underlying DBMS to efficiently enumerate the capabilities ascribed to a given user. We also present an evaluation of a prototype implementation of our framework that demonstrates the practicality of our approach. As CTM extends the *RT* family of trust management languages—which are representative of a large class of Datalog-based trust management systems—our work is likely generalizable to other trust management approaches.

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

General Terms: Security

Keywords: Credentials, databases, policy, reputation, trust

1. INTRODUCTION

Today's organizations bear little resemblance to the highly centralized monoliths of old, having instead embraced the pervasiveness and reliability of the Internet to become ever more distributed entities. For example, it is the norm for an organization to be comprised of many logically and physically decentralized divisions acting autonomously, companies are increasingly forming virtual organizations to carry out a range of short- and long-term goals, and the rise of Web 2.0 has involved users in content development and dissemination like never before. Unfortunately, legacy identity-based access control technologies are ill-suited to meet the needs of these emerging dynamic organizations.

As a result, the past decade has seen much research on trust management [2,3,5,20], trust negotiation [19,27,29,30], and distributed proof construction [1, 12, 15, 24] approaches to authorization. More recently, researchers have begun to combine these types of systems with the decentralized reputation and recommendation systems used in P2P and social networks [4, 7, 9, 10, 18] to take a more comprehensive approach to managing trust. These types of trust management systems allow administrators to manage and query the protection state of a distributed system in a manner that leverages—rather than works around—the decentralized nature of the system.

In order to be a viable solution for emerging networked environments, a trust management system must be capable of *efficiently* answering queries over the protection state of the environment, even when handling policies with rich structures and complex reputation functions. Existing research [1,2,6,8,17,23] has done an admirable job of addressing the proof of compliance question (*Can user u satisfy policy p?*), but relatively little attention has been given to the policy satisfaction question (*Which users can satisfy the policy p?*) and the capability question (*What permissions does user u have?*). The latter questions are intrinsically difficult to solve in many trust management systems, but are necessary if administrators are to fully understand the protection state of the system. Furthermore, efficiency is not the only concern when considering the deployment of trust management systems in commercial environments: these systems must also be deployable on top of existing IT infrastructures. To this end, De Capitani Di Vimercati et al. show that a bare-bones trust management system can be implemented entirely within a DBMS [11]. While this is a step in the right direction, it says little about the adoption of the myriad logical trust management systems proposed in the research literature.

In this paper, we further pursue this line of work by developing an implementation of the composite trust management system CTM [18] that is capable of efficiently answering proof of compliance, policy satisfaction, and capability queries by leveraging both logical inference, as well as an open-source DBMS substrate. CTM extends the *RT* family of role-based logical trust management systems [20] with support for aggregate horizontal trust calculations (e.g., reputation functions or QoS metrics). Because *RT*—and therefore CTM—is representative of a broad class of logical trust management systems and has been well-studied in the research literature, our work takes a step beyond [11] to show that we need not resort to special-purpose solutions to realize the benefits of trust management within commercial IT infrastructures. In addressing this challenge, this paper makes the following contributions:

- We describe the design of a framework for trust management policy management built upon a commercial relational DBMS substrate. This framework automatically synchronizes an organization's logical policy ruleset with the underlying database, and uses these complimentary views of an organization's protection state to efficiently answer many types of trust management queries.
- Within the context of this framework, we describe a correct and complete compilation procedure for transforming a set of CTM policy credentials into a collection of dynamic views over a standard relational database. The resulting database provides an efficient mechanism for answering proof of compliance and policy satisfaction queries.
- We present a novel algorithm for answering capability review queries over a composite trust management system; i.e., given a particular user, enumerate the set of rights possessed by this user. Our algorithm takes advantage of our framework's logical rule set to forward-chain from the user's set of base rights whenever possible, while simultaneously leveraging the database's ability to quickly aggregate horizontal trust assessments and perform complex joins over role memberships to answer more difficult membership queries. This hybrid forward-chaining/top-down processing approach increases the efficiency with which capability review queries can be answered.
- To assess the efficiency of our framework, we present a comprehensive evaluation of a prototype implementation. In particular, we describe several microbenchmarks that gauge the core costs involved with processing queries over basic trust management policies. We then discuss the results of evaluating a larger workload that simulates the use of our framework within the context of a set of hierarchically organized virtual organizations.

When viewed collectively, the above contributions provide further insight into questions surrounding the deployment of trust management systems. First, we demonstrate that even complex CTM policies can be evaluated efficiently in centralized systems, a challenge that was left as an open problem in [18]. Second, our framework shows that existing commercial technologies form a viable substrate for the realization of feature-rich trust management in complex systems.

The rest of this paper is organized as follows. Section 2 provides a brief introduction to the CTM composite trust management policy language, and then discuss the types of queries that a trust management framework should be capable of efficiently answering. In Section 3, we describe the design of a centralized framework for processing CTM queries using off-the-shelf relational DBMS technologies. Section 4 describes how this framework can be used to efficiently process trust management queries. In particular, we present a compilation procedure for transforming a collection of CTM policy credentials into a set of dynamic views over a relational database, and describe a novel hybrid algorithm for processing capability review queries over trust management policies. In Section 5 we present a comprehensive performance evaluation of our framework. Finally, Section 6 presents an overview of related research efforts, while Section 7 discusses our conclusions and promising directions for future research.

2. BACKGROUND

To make our discussion concrete, in the rest of this paper we give the specific design of the framework to support CTM, an extension of the *RT* family of logical trust management languages. One distinguished feature of CTM is its support for aggregate trust evaluation (e.g., reputation functions) and the composition of credential-based and reputation-based trust. This feature, on the one hand, significantly improves the expressiveness of trust management languages. On the other hand, it imposes challenges to efficient trust policy evaluation. In this section, we give a brief overview of CTM, and discuss the types of queries that we expect a trust policy evaluation engine to answer efficiently.

2.1 Composite Trust Management

Two types of trust affect human interactions in a society: vertical trust and horizontal trust. Vertical trust is a trust relationship between individuals and institutions or authorities (often in the form of credentials). Horizontal trust captures the trust that can be inferred from the observations and opinions of other peers (often in the form of reputations aggregated from individual's experience). These two types of trust are complementary and often used in concert during one's decision making. Lee and Yu proposed CTM, a composite trust management language to support flexible composition of vertical and horizontal trust [18]. The design of CTM is built on *RT* with extensions to support constraints on reputation evaluations.

As in *RT* and many other trust management languages, CTM uses the concept of role membership to define a set-based semantics for policies. Intuitively, a role defines a set of principals possessing the same properties, while a policy defines a set of role memberships that must be possessed by an authorized principal. Like in *RT*, principals in CTM are identified by means of identity certificates. A role is defined simply as strings identifying the name of the role. Policy statements are expressed as one or more of these role definitions and are encoded as role definition credentials signed by the author of the role definition. CTM supports the following four basic types of role definitions in *RT*.

Simple Member. A role definition of the form $K_A.R \leftarrow K_D$ encodes the fact that principal K_A considers principal K_D to be a member of the role $K_A.R$. For example, $StateU.student \leftarrow Alice$ says that Alice belongs to the role $StateU.student$.

Simple Containment. A role definition of the form $K_A.R \leftarrow K_B.R_1$ encodes the fact that principal K_A defines the role $K_A.R$ to contain all members of the role $K_B.R_1$, which is defined by principal K_B. For example, $eBook.preferred\_customer \leftarrow StateU.student$ says that StateU students are also preferred customers of eBook, an online book store. This is a typical way to specify delegation in decentralized systems.

Linking Containment. A role definition of the form $K_A.R \leftarrow K_A.R_1.R_2$ is called a linked role. This defines the members of $K_A.R$ to contain all members of $K_B.R_2$ for each K_B that is a member of $K_A.R_1$. For example, $eBook.preferred\_customer \leftarrow ABU.accredited\_univ.student$ says that any student of a university that is accredited by ABU (Accreditation Board for Universities) is a preferred customer of eBook.

Intersection Containment. A role definition of the form $K_A.R \leftarrow K_{B_1}.R_1 \cap \cdots \cap K_{B_n}.R_n$ defines $K_A.R$ to contain the principals who are members of each role $K_{B_i}.R_i$ where $1 \leq i \leq n$. For example, $eBook.preferred\_customer \leftarrow StateU.student \cap ACM.member$ says that an StateU student who is also a member of ACM is a preferred customer of eBook.

To support horizontal trust generally, CTM assumes principals interact through a series of transactions. Feedbacks are issued by involved principals after a transaction completes. A feedback may contain a variety of attributes, including the issuer (i.e., the principal who issues the feedback), the signer (i.e., the principal who certifies the feedback), the subject (i.e., about whose behavior the feedback applies to), a single rating, and other transaction-specific properties. A horizontal trust evaluation thus can be modeled as the application of a trust function $f : 2^{\mathcal{F}} \times \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{R}$, where $\mathcal{F}$ is a set of feedbacks about transactions among principals in a system, $\mathcal{P}$ is the set of principals, and $\mathcal{R}$ is the reputation domain (i.e., the possible trust values for reputation-based trust). Intuitively, given a principal A and B, and a set of feedbacks F, $f(F, A, B)$ returns A's trust assessment of B based on some evidence, i.e., the set of feedbacks in F. Here A and B are called the source and the target of a reputation evaluation respectively.

Given the above formalism, CTM allows a domain to define roles in terms of reputation evaluation. Note that a domain may impose flexible constraints on the evaluation of a reputation function, including the set of feedbacks fed to the reputation function. Specifically, besides the above four types of roles as defined in *RT*, CTM further supports the concept of *aggregate containment roles.* For ease of presentation, we consider the following simplified definition of aggregate containment:

Aggregate Containment. Let $\diamond$ represent a comparison operator (e.g., $<, \leq, =, \geq, >$, or $\neq$). The role definition $K_A.R \leftarrow K_B.f(issuer = K_i.R_i, output \diamond c_o)$ defines the role $K_A.R$ to contain all principals whose horizontal trust level satisfies the constraint $output \diamond c_o$ after principal K_B invokes the horizontal trust assessment function f when considering feedback reports issued by principals in the role $K_i.R_i$. For example, $Alice.R_1 \leftarrow Alice.f(issuer = Alice.Friend, output \geq 0.8)$ defines that a principal B is a member of $Alice.R_1$ if Alice's trust evaluation of B using the function f is over 0.8, when considering feedback reports issued by Alice's friends.

As is the case for the four basic types of roles in *RT*, CTM has a well-defined set-theoretic semantics for aggregate containment roles. Therefore, they can be flexibly combined with other types of roles, which enables arbitrary composition of vertical and horizontal trust. The above example in fact shows how other roles are used in the definition of an aggregate containment role. Similarly, aggregate containment roles may also be used in the definition of vertical roles. The full extent of this potential to arbitrarily compose horizontal and vertical trust is discussed in detail in [18]; as such we do not elaborate upon it further in this paper

2.2 Trust Query Types

Many security vulnerabilities in information systems are caused by the misconfiguration of security policies, which often results in either giving excessive or insufficient privileges for principals to carry their tasks. Privilege review is one of the key steps to discover and fix such configuration mistakes. It is in particular important to decentralized trust management because one domain's trust decision may affect privileges of many principals globally. A trust management system should not only have the capability to answer access control decisions, but also have efficient support for various privilege review requirements.

In this paper we consider the support for the following three useful types of privilege review queries against a trust management system.

Proof of compliance query. Determine whether a principal is a member of a role. Such queries often correspond to access control decisions when a principal requests access to certain resources. They are frequently issued and should be answered not only correctly but very efficiently.

Role membership query. Determine all the members of a role. Such queries are useful to check to what extend certain privileges have been propagated in a system. They are also often needed for "what-if" type of analysis to determine the effect of a change of policy.

Capability query. Given a principal, determine all the roles of which the principal is a member. Such queries are useful to check whether a user possess excessive privileges.

Though role membership queries and capability queries do not happen as frequently as proof of compliance queries, it is still desirable to answer them with reasonable efficiency. Similar to the implementation of the access matrix in traditional access control, how role information is maintained in a trust management system greatly affects the efficiency of query answering.

3. THE DESIGN OF A TRUST EVALUATION ENGINE

A trust evaluation engine is essentially a reference monitor that determines whether a principal possesses the necessary privileges to perform certain actions or access certain resources. In CTM, as privileges are implicitly wrapped as roles, a trust evaluation engine ensures the correct mapping between users and roles. It also helps to answer policy analysis queries as mentioned in the previous section.

Instead of developing a trust evaluation engine as a stand alone software system from scratch, in this paper we advocate to build such a system on top of commodity relational database management systems (DBMSs), which we believe brings at least the following benefits:

1. **Easy deployment.** DBMSs are arguably one of the most widely deployed and managed software systems in enterprises, organizations, and governments. By building on top of DBMSs, a trust evaluation engine can easily be integrated with existing IT infrastructures.

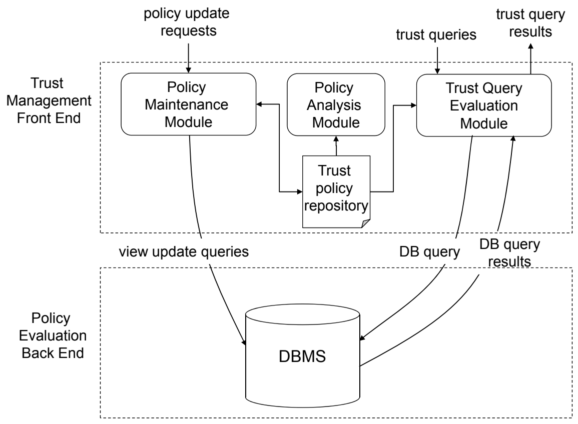


Figure 1: The design of a trust evaluation engine built on top of database management systems

2. **Strong support from DBMSs.** Many key features of trust management systems can be mapped to DBMS concepts in a straightforward manner. For example, simple membership credentials for the same role can be naturally grouped together into a relation (table) in a database. Further, if parameters are involved for roles, they can be easily added as attributes of a relation.

   Trust management systems can also directly benefit from the mature data processing techniques used in DBMSs. For example, DBMSs are optimized to handle complicated aggregations over large scale data sets, which is ideal for supporting horizontal trust evaluation (e.g., reputation calculations). Furthermore, if a horizontal trust calculation involves an aggregation operation that is not supported natively, most DBMSs allow these types of functions to be easily integrated via user-defined aggregation functions.

   DBMSs provide rich indexing techniques that can be readily utilized to boost the performance of trust query answering. Further, by treating each trust query and trust policy update as a transaction, a DBMS guarantees their atomic execution, which ensures the correctness of trust queries, as well as the overall *consistency* of the system view used to evaluate the query [16], even when multiple queries and updates are processed concurrently.

3. **Lightweight trust evaluation engine.** Because of strong support from DBMSs, a large portion of the trust query answering can be shifted to the DBMSs. The actual trust evaluation engine itself thus is lightweight, and easily verified for correctness.

In [11], De Capitani Di Vimercati et al. proposed to use a DBMS as a repository of credentials, and designed a SQL-like language to store and retrieve credentials to and from a credential database. The goal is to have a trust management system completely implemented in a DBMS. However, we argue that not all the trust management functionalities are suitable to be implemented in DBMSs. For example, a security officer often needs to perform policy analysis to verify that a trust policy preserves certain security properties such as safety and availability. On the one hand, these types of analyses are difficult to perform purely by a DBMS. On the other hand, using a DBMS as only a credential store does not fully utilize its data processing power. In this paper, we propose to strike a balance between these two extremes by using a DBMS as a back-end processing system to facilitate the evaluation of trust queries.

Figure 1 shows a high-level view of our proposed system architecture, which is composed of two distinct layers: a *logical* trust management front end and a *relational* query processing back end. The trust management front end is the main interface through which users interact with the trust management system. This layer is comprised of three modules: the policy maintenance, the policy analysis, and the query evaluation modules. The policy maintenance module allows administrators to manage the logical ruleset parameterizing the system, and is responsible for synchronizing with the relational back end. The trust query evaluation module is used to process the types of trust queries described in Section 2.2 by interacting with the logical ruleset and the relational back end. Lastly, the policy analysis module is responsible for all other logical policy analysis tasks (e.g., safety analysis [21]).

The relational back-end database does not store copies of the logical credentials themselves, but rather manages auxiliary information corresponding to an *interpretation* of those credentials within the context of trust query processing. Specifically, it maintains two tables—`base_roles` and `reports`—that hold role memberships defined by simple membership credentials ($A.R \leftarrow B$) and feedback reports issued by users in the system, respectively. The key insight that we leverage is that all other forms of CTM credentials represent, in essence, *queries* that define one role in terms of other roles in the system. To this end, the policy management module in the logical front end compiles these credentials into *dynamic views* in the relational back end that select the members of a given role by querying the membership sets of other roles. As we will see in the next section, such a relational back end greatly simplifies the processing of proof of compliance and role membership queries, while a combined logical/relational approach can be used to efficiently answer capability queries.

Note that the relational query processing back end does not require any change to the DBMS. In other words, any commodity DBMS can be used directly with the logical trust management front end to form a trust policy evaluation engine.

4. QUERY PROCESSING

In this section, we elaborate on how our framework can be used to answer each of the three query types described in Section 2.2. We first describe a process through which CTM credentials can be compiled into dynamic views in a DBMS, which facilitates efficient processing of *proof of compliance* and *role membership* queries. We then present the details of a hybrid algorithm that uses forward chaining and top-down query processing to efficiently execute *capability* queries. For simplicity, in this paper we do not consider circular dependent roles; such types of role hierarchy are extremely uncommon in practice.

4.1 Compiling CTM Credentials

As was described previously, the `base_roles` and `reports` tables of the DBMS are populated with the role memberships described by simple membership credentials and the

```
CREATE TABLE base_roles(
  owner   varchar(30), # Pricipal defining the role
  role    varchar(30), # Name of the role being define
  subject varchar(30)  # Principal defined as a role member
);

CREATE TABLE reports(
  issuer varchar(30),  # Issuer of the feedback report
  target varchar(30),  # Target of the feedback report
  rating double,       # Single rating value
  date   date          # Date that the report was issued
};
```

Figure 2: Schemas defining the `base_roles` and `reports` tables.

feedback reports issued by principals in the system, respectively. To simplify the presentation in this section, we assume the use of only unparameterized CTM credentials and feedback reports consisting of a single value. As a result, the `base_roles` and `reports` tables can be defined by the schemas shown in Figure 2. For a simple membership credential $A.R \leftarrow B$, a row will be entered into the `base_roles` table identifying A as the role owner, R as the role, and B as a subject. Similarly, if principal A generates a feedback report about principal B, the corresponding row in the `reports` table will identify A as the issuer of the feedback report and B as the target. Section 4.2 explains how the techniques detailed in this section can be extended to support parameterized CTM or RT_1 credentials.

We now describe how the set C of intersection containment, linking containment, and aggregate containment credentials can be compiled into a collection of dynamic views defined over these base tables, as well as other views defined during the compilation procedure. In our discussion, we treat a simple containment credential as an intersection containment with only one role in its body. We first make two assumptions:

- The logical component of our framework stores the set C of CTM credentials as an ordered list, *cList*. This list of credentials is arranged such that for any two credentials c_i and c_j, c_j can be defined in terms of c_i if and only if $j > i$. That is, *cList* is arranged such that if one role depends on other roles, it is defined after its dependencies are defined.
- We have access to a data structure *roleManagers* : *String* $\rightarrow 2^P$ that maps a role name to the set of principals defining this role. For instance, if *AliceInc* and *BobCorp* both define an *employee* role, *roleManagers* will map the role *employee* to the set of principals $\{AliceInc, BobCorp\}$.

Given the $cList = \langle c_1, ..., c_n \rangle$ and *roleManagers* data structures, the compilation of CTM credentials into dynamic views proceeds according to the following $O(n)$ procedure:

1. Create a map *viewDefs* : *String* $\rightarrow$ *String*. This structure will be used during the compilation process to associate a role name (e.g., $A.R$) with the SQL view definition command that will eventually be inserted into the database.
2. For each distinct role $A.R$ defined in the `base_roles` table, generate the following SQL view definition, execute it, and associate it with $A.R$ in the *viewDefs* map:

```
CREATE OR REPLACE VIEW A_R(subject) AS
  SELECT subject FROM base_roles WHERE owner='A' AND role='R'
```

   This view definition ensures that any query to the view `A_R` will consider the role memberships asserted using simple membership credentials.

3. For each role definition credential c_i:

   - If c_i is an intersection containment credential of the form $A.R \leftarrow B_1.R_1 \cap \cdots \cap B_n.R_n$, generate the following SQL selection statement s_i:

```
SELECT subject FROM B1_R1, ..., Bn_Rn WHERE
  B1_R1.subject = B2_R2.subject AND
  ...
  B1_R1.subject = Bn_Rn.subject
```

     This selection gathers all principals who are members of each role $B_1.R_1, \ldots, B_n.R_n$.

   - If c_i is a linking containment credential of the form $A.R \leftarrow A.R_1.R_2$, use the *roleManagers* map to look up the set $\{B_1, \ldots, B_n\}$ of principals defining an R_2 role. Then, generate the following SQL statement v_i defining the intermediate view `R2`:

```
CREATE OR REPLACE VIEW R2(owner, subject) AS
  SELECT 'B1' AS owner, subject FROM B1_R2 UNION
  ...
  SELECT 'Bn' AS owner, subject FROM Bn_R2;
```

     This intermediate view can then be used to define the following SQL selection statement s_i:

```
SELECT DISTINCT R2.subject FROM A_R1, R2 WHERE
  A_R1.subject = R2.owner
```

     Essentially, this selection finds all members of $X.R_2$ such that X is a member of $A.R_1$. As was discussed in Section 2.1, this is exactly the semantics of the role definition $A.R \leftarrow A.R_1.R_2$.

   - If c_i is an aggregate containment credential of the form $A.R \leftarrow B_1.f(issuer = B_2.R_2, output \geq c)$, generate the following SQL selection s_i:

```
SELECT target FROM reports, B2_R2
  WHERE reports.issuer =  B2_R2.subject
  GROUP BY target HAVING f(rating) > c
```

     The query s_i uses the function f—which is executed as a stored procedure within the DBMS—to aggregate the feedback reports issued about a particular target user, provided that these reports were issued by a member of the role $B_2.R_2$. Note that *only* users whose aggregate feedback score is above the threshold c are eventually selected.

   At this point, look up definition d_i in the *viewDefs* map that corresponds to the role $A.R$. If d_i is undefined, set d_i = `CREATE OR REPLACE VIEW A_R(subject) AS` $\oplus\ s_i$, where $\oplus$ denotes string concatenation. Otherwise, set $d_i = d_i \oplus$ `UNION` $\oplus\ s_i$. If a temporary view command v_i was generated, set $d_i = v_i \oplus d_i$. Finally, execute the updated command d_i and store it back in the *viewDefs* map.

Intuitively, the above process provides a bottom-up means of compiling CTM credentials into a relational database representation. Since simple membership credentials and individual feedback reports have no dependencies on any other data items, they can be inserted into `base_roles` and `reports` tables straight away. After this basic level of data

```
CREATE OR REPLACE VIEW AliceInc_employee(subject) AS
  SELECT subject FROM base_roles WHERE owner='AliceInc'
  AND role='employee';
CREATE OR REPLACE VIEW BobCorp_employee(subject) AS
  SELECT subject FROM base_roles WHERE owner='BobCorp'
  AND role='employee';
CREATE OR REPLACE VIEW BBB_member(subject) AS
  SELECT subject FROM base_roles WHERE owner='BBB'
  AND role='member';
CREATE OR REPLACE VIEW ACM_member(subject) AS
  SELECT subject FROM base_roles WHERE owner='ACM'
  AND role='member';

CREATE OR REPLACE VIEW BBB_goodRep(subject) AS
  SELECT target FROM reports, ACM_member
  WHERE reports.issuer =  ACM_member.subject
  GROUP BY target HAVING rep(rating) > c;

CREATE OR REPLACE VIEW ePub_trusted(subject) AS
  SELECT subject FROM BBB_member, BBB_goodRep WHERE
  BBB_member.subject = BBB_goodRep.subject;

CREATE OR REPLACE VIEW employee AS
  SELECT 'AliceInc' AS owner, subject FROM AliceInc_employee UNION
  SELECT 'BobCorp' AS owner, subject FROM BobCorp_employee;
CREATE OR REPLACE VIEW ePub_discount AS
  SELECT DISTINCT employee.subject FROM ePub_trusted, employee
  WHERE ePub_trusted.subject = employee.owner;
```

Figure 3: A compiled version of the CTM policy described in Section 4.1

has been entered into the database, Step 2 generates a new view `A_R` for each role $A.R$ described by the `base_roles` table. Step 3 compiles each credential in *cList* into an SQL selection over previously-defined views. The sort invariant on *cList* ensures that only references to previously-defined views are required as view definitions are inserted or updated. As a concrete example, consider the following CTM credentials:

$$\begin{aligned} \mathit{BBB.goodRep} &\leftarrow \mathit{BBB.rep}(\mathit{issuer} = \mathit{ACM.member}, \\ &\qquad\qquad \mathit{output} > 0.9) \\ \mathit{ePub.trusted} &\leftarrow \mathit{BBB.member} \cap \mathit{BBB.goodRep} \\ \mathit{ePub.discount} &\leftarrow \mathit{ePub.trusted.employee} \end{aligned}$$

This policy asserts that ePub is willing to give a discount to employees of trusted organizations. ePub considers an organization trusted if the organization is a member of the BBB and is rated highly by members of the ACM. Assuming that AliceInc and BobCorp each define an employee role in the server's `base_roles` table, Figure 3 describes how the above collection of CTM credentials can be compiled and represented in a DBMS.

Given this relational representation of a set of CTM credentials, answering the *proof of compliance* and *role membership* queries becomes trivial. Specifically, determining all members of a role $A.R$ is a simple matter of executing the query `SELECT subject FROM A_R`. Similarly, determining whether Alice is a member of the role $A.R$ can be answered by executing the query `SELECT subject FROM A_R WHERE subject='Alice'`; a non-zero recordset indicates that Alice is a member of $A.R$, while an empty recordset indicates that she is not. This leads us to the following theorem:

Theorem 1 (Correctness & Completeness). *Let cList be an ordered collection of CTM credentials, and let DB be the database resulting from compiling cList using the above process. DB finds the set U of users belonging to a role $A.R$ if and only if there exist CTM proofs of compliance demonstrating that u is a member of $A.R$ for each $u \in U$.*

Theorem 1 can be proved using a relatively straightforward, albeit lengthy, structural induction that demonstrates a 1-to-1 correspondence between the views generated by our compilation procedure and the set theoretic semantics of CTM credentials. In the interest of space, we omit the details of the proof in this paper.

4.2 Supporting Parameterized Roles

We now provide a brief intuition for how the compilation procedure described in Section 4.1 can be adapted to support parameterized CTM role definitions. First, we assume that the maximum number of parameters to any given role can be bounded by some integer n. Then, the schema for the `base_roles` table can be defined as follows:

```
CREATE TABLE base_roles(
  owner   varchar(30), # Pricipal defining the role
  role    varchar(30), # Name of the role being define
  subject varchar(30), # Principal defined as a role member
  int1    int,         # Integer parameter 1
  ...
  intn    int,         # Integer parameter n
  str1    varchar(30), # String parameter 1
  ...
  strn    varchar(30), # String parameter n
  double1 double,      # Double parameter 1
  ...
  doublen double       # Double parameter n
);
```

Given such a definition for the `base_roles` table, we can modify the compilation procedure from Section 4.1 to generate parameterized views. We now present a simple example to demonstrate the intuition behind this revised approach. Consider the following role definition:

$$\begin{aligned} &\mathit{History.Trust}(\mathit{area} = \text{“}\mathit{Tech}\text{”}) \leftarrow \\ &\qquad \mathit{StateU.faculty}(\mathit{since} \geq 2006) \cap \\ &\qquad \mathit{AandS.rep}(\mathit{committee} = \text{“}\mathit{Technology}\text{”}) \end{aligned}$$

The above role definition says that the History department trusts faculty members at StateU with respect to “Tech” questions if the faculty member is an Arts and Sciences representative on the technology committee and has been a faculty member since at least 2006. Assuming that roles have at most one parameter, the above definition can be compiled into the following dynamic view:

```
CREATE OR REPLACE VIEW
  History_Trust(subject, int1, str1, double1) AS
  SELECT subject, "Tech" as str1, 0 as int1, 0.0 as double1
  FROM StateU_faculty, AandS_rep
  WHERE StateU_faculty.subject = AandS_rep.subject AND
        StateU_faculty.int1 >= 2006 AND
        AandS_rep.str1 = "Technology";
```

Note that each view must define columns for each available parameter, and that view definitions may constrain the parameter columns of the views that they query. Similar adaptations can be made to all compilation rules defined in Section 4.1.

Algorithm 1 Algorithm for resolving capability queries.

```
 1: Function CapQuery($List\langle Credential\rangle cList$, $User\ u$) : $Set\langle Role\rangle$
 2: // Query database to determine base role memberships
 3: $Set\langle Role\rangle\ roles$ = Exec(SELECT CONCAT(owner, '.', role) FROM base_roles WHERE subject = $u$)
 4: $Set\langle String\rangle\ roleNames = \emptyset$
 5: for all $A.R \in roles$ do
 6:    $roleNames.insert(R)$
 7:
 8: // Process our order-sorted list of CTM credentials
 9: for all $c \in cList$, where $c$ is of the form $A.R \leftarrow \ldots$ do
10:    if $A.R \notin roles$ then
11:       // Forward-chain over simple- and intersection-containment credentials
12:       if ($c$ is of the form $A.R \leftarrow B_1.R_1 \cap \cdots \cap B_n.R_n$) $\wedge$ ($\{B_1.R_1, \ldots, B_n.R_n\} \subseteq roles$) then
13:          $roles.insert(A.R)$
14:          $roleNames.insert(R)$
15:       // Use the DB to process linking- and aggregate-containment queries
16:       if ($c$ is of the form $A.R \leftarrow A.R_1.R_2 \wedge R_2 \in roleNames$) $\vee$ ($c$ is an aggregate containment) then
17:          $int\ count$ = Exec(SELECT COUNT(*) FROM A_R WHERE subject='$u$')
18:          if $count > 0$ then
19:             $roles.insert(A.R)$
20:             $roleNames.insert(R)$
21:
22: return $roles$
```

4.3 Answering Capability Queries

The hierarchical collection of dynamic views generated by the compilation procedure described in Section 4.1 is well-suited for answering proof of compliance and role membership queries precisely because it takes a role-centric view of the system. That is, views are explicitly created to manage role memberships. This is a reasonable design choice, since these types of queries are the most often executed in practice: proof of compliance queries are issued to check permissions at each resource access request, and role membership queries can be used to materialize a concrete ACL for a resource. Unfortunately, this organization is sub-optimal for answering capability queries. Specifically, to determine the set of roles to which a user A belongs, a proof of compliance query would need to be issued for each individual role! This is clearly unacceptable.

To address this issue, we have developed a novel algorithm for answering capability queries that leverages both the logical ruleset stored by our framework, as well the dynamic views maintained within the DBMS. At a high level, our approach uses the logical ruleset to forward-chain whenever possible, while relying on the database to answer role membership queries that cannot be established efficiently via forward-chaining. This is similar in spirit to, though intrinsically different than, the bidirectional search algorithm proposed in [23]. Algorithm 1 illustrates this approach in detail.

Algorithm 1 takes two parameters: the sorted list of CTM credentials *cList*, and a user u. The algorithm begins by querying the database to gather the base permissions assigned to the user u and then populating the *roles* and *roleNames* sets. These sets track the roles that a user is a member of, and are assumed to be implemented using a data structure with $O(1)$ insert and lookup (e.g., a Java HashSet). The algorithm then iterates over *cList*, skipping any credentials that define roles to which the user is already a member. If the current role c defines a simple- or intersection containment, the algorithm forwards chains, using *roles* to determine whether the user has the prerequisite roles to establish membership in the role defined by c. If so, the *roles* and *roleNames* sets are updated accordingly and the algorithm moves on to the next credential in *cList*.

We note that forward chaining cannot be used to establish membership in roles defined by linking containment credentials without also forward chaining the role memberships of (potentially) all other principals in the system. For example, the linking containment $A.R \leftarrow A.R_1.R_2$ requires information not only about u's membership in $X.R_2$, but also about X's membership in $A.R_1$ (for *all* such X!). Similarly, forward chaining cannot be used to aggregate feedback reports. As such, line 16 acts as a guard condition determining whether the database should be used to process a particular role membership in a top-down manner. Note that a linking containment credential $A.R \leftarrow A.R_1.R_2$ is only processed if u is a member of some $X.R_2$ as indicated by the *roleNames* set: if there is not a member of some $X.R_2$, then they are certainly not a member of $A.R_1.R_2$. Finally, we have the following theorem:

Theorem 2 (Correctness & Completeness). *Let cList be the set of CTM roles parameterizing the system, u be a user, and R be the set of roles returned by the function call* CapQuery($cList, u$). *The set R of roles represents* exactly *the set of role memberships held by u.*

Proof. (Sketch) By Theorem 1, we know that the query on line 3 will always return precisely the base set of permissions assigned to u. Given the order-sort invariant imposed on *cList*, we can prove by induction that the forward chaining procedure described on lines 12–14 will determine that $u \in \mathsf{member}(A.R)$ if and only if $u \in \mathsf{member}(B_i.R_i)$ for each prerequisite B_i. Further, Theorem 1 gives us that the top down processing procedure described on lines 16–20 will always accurately determine u's membership in roles defined by linking or aggregate containment credentials. So for all $A.R$ defined in *cList*, u's membership is accurately assessed by CapQuery($cList, u$). $\square$

This combined logical/relational view of a CTM policy provides us with a significant speedup in processing capability queries, since (i) forward-chaining is extremely efficient and (ii) we can leverage prior knowledge regarding role membership to issue only those database queries that have a chance at succeeding. Specifically, the brute-force approach to capability review requires $O(N)$ proof of compliance queries, where $N = |cList|$. On the other hand, Algorithm 1 requires $O(p_{la}N)$ queries to the database, where p_{la} is the percentage of credentials in *cList* defining linking- or aggregate containments. As we will see in the next section, the efficiency gains realized by Algorithm 1 are quite noteworthy in practice.

5. EXPERIMENTAL EVALUATION

In this section, we discuss the performance overheads associated with trust management query processing within a research prototype implementation of our proposed framework. We first explore several microbenchmarks that illustrate the relative costs of evaluating role membership and proof of compliance queries over roles defined by each type of CTM credential. We then present a more detailed virtual organization scenario, within which we evaluate the costs proof of compliance, role membership, and capability queries in a more meaningful setting. Finally, we conclude this section by discussing the trade-offs associated with leveraging materialized, rather than dynamic, views within a DBMS-based trust management system.

5.1 Experimental Setup

All experiments discussed in this section were carried out on a machine equipped with a 2.1 GHz AMD Opteron processor and 4 GB of RAM, running Red Hat Enterprise Linux WS release 4. The core components of our framework—including the policy compiler, logical credential management facilities, and query processing algorithms—were written in Java and compiled using the Sun Java JDK version 1.6.0_14. A stock installation of MySQL version 5.1.35 was used as our back-end database. All reputation aggregation functions were written as user-defined functions in C and loaded by MySQL as shared objects.

5.2 Microbenchmarks

To examine the relative costs of evaluating proof of compliance and role membership queries in a controlled setting, we first initialized three databases: small (100 users, 10 feedback reports per user), medium (500 users, 20 feedback reports per user), and large (1000 users, 30 feedback reports per user). We then defined 50 roles in the `base_roles` table and made each user a member of each role. When then compiled roles defined by the following types of CTM credentials into dynamic views in each database:

- Linear chains of delegation of up to length 5 specified by simple containment credentials.
- Binary trees of up to height 4 specified by intersection containment credentials.
- Linking containments of the form $A.R_1.R_2$ defined such that the members of $A.R_1$ could be determined by using the `base_roles` table and the $X.R_2$ role is a linear chain of delegation with height varying from 1 to 5.
- Aggregate containments whose issuer filter role is specified by a linear chain of delegation with height varying from 1 to 5.

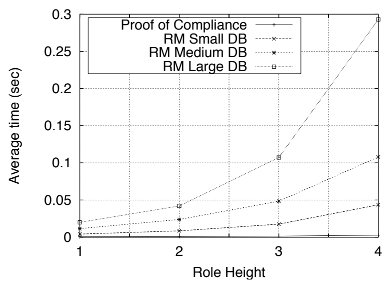
The result of our experiments are depicted in Figure 4. Figure 4(a) shows that the cost of evaluating intersection containment credentials increases exponentially in the height of the credential. This is expected, as each increase in height requires evaluating the member sets of twice as many base roles. Note that across all database sizes, proof of compliance remained very efficient, despite the increased costs of enumerating all role members. In Section 5.4, we will discuss some potential avenues for combating these types of increases. Figures 4(b) and (c) describe the costs of evaluating queries about linking containment and aggregate containment roles. In both cases, the cost does *not* increase with the height of the height of the linear delegation used, as the cost of evaluating linear chains of delegation is negligible. Further, processing proof of compliance and role membership queries takes approximately the same amount of time due to the joins required to process these queries.

5.3 A Virtual Organization Scenario

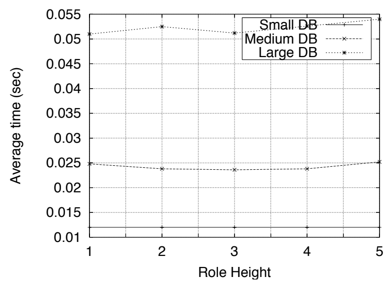
While the above microbenchmarks illustrate the relative costs of querying different types role definitions, they say very little about the costs of analyzing policies containing large number of credentials, mixed hierarchies of credentials, or roles defined by multiple credentials. To find the answers to these questions, we developed a tool that generates synthetic hierarchical CTM policies within the context of a set of collaborating virtual organizations (VOs). The parameters to our tool are similar to those described in the role-mining literature (e.g., see [25, 26]), and are described in Table 1. In this paper, we generate the base role assignments for three companies, and then build four layers VOs on top of these companies. Each layer contains three VOs, each of which defines its roles by generating CTM credentials that reference the roles defined by the three VOs (or companies) one level below it.

Within this context, we examined the cost of processing queries across a range of database sizes and CTM rule complexities. In particular, we considered small (100 users/ company, 10 feedback reports/user), medium (500 users/ company, 20 feedback reports/user), and large (1000 users/ company, 30 feedback reports/user) databases, as well as low (100% intersection containment), medium (90% intersection containment, 5% linking containment, 5% aggregate containment), and high (70% intersection containment, 15% linking containment, 15% aggregate containment) complexity rule sets. In all cases, each company defined 30 roles using simple memberships, while each VO defined 30 roles according to the other parameters described in Table 1.

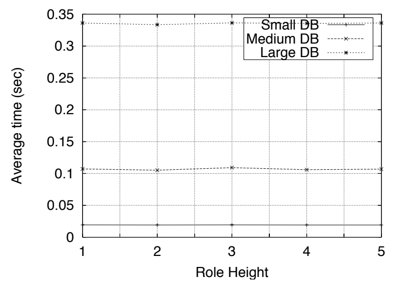
Figure 5(a) describes how the time required to answer proof of compliance queries varies over medium complexity databases of various size. Figure 5(b) shows how the cost of answering proof of compliance queries in a medium sized database is affected by varying CTM rule complexities. In both cases, we see that changes to the size or complexity of the database have an observable—but not tremendous—impact on query execution time. Rather, the variable that most influences query runtime is the *height* at which a role is defined: e.g., *all* queries at levels 2 and below run extremely quickly, regardless of how big (resp. complex) the underlying database (resp. CTM ruleset) is, while higher-level queries



(a) Intersection containment

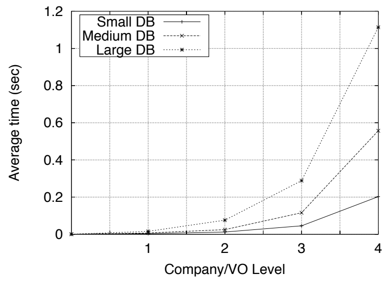


(b) Linking containment

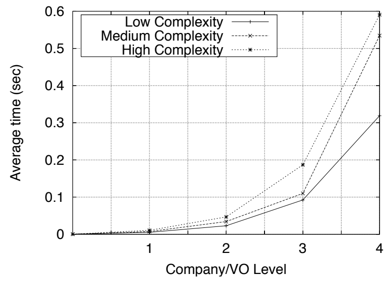


(c) Aggregate containment

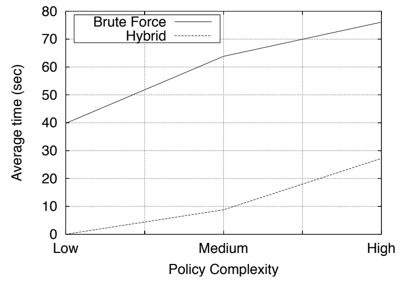
Figure 4: Performance results from microbenchmark scenarios.



(a) Varied database complexity



(b) Varied rule complexity



(c) Capability query

Figure 5: Performance results from evaluation within the VO scenario.

are much more expensive. This is an artifact of the "bushy tree" dataset that we generated, as queries at level n can be defined using up to 3 CTM credentials, each of which may intersect up to 3 roles at level $n-1$. In practice, we would expect to see far less bushy role structures.

Figure 5(c) describes the average time required to execute capability queries over medium sized databases of low, medium, and high complexity. The brute force approach of carrying out one proof of compliance query for each possible role in the database is compared against the algorithm presented in Section 4.3. As expected, our algorithm outperforms the brute force approach in all cases. In particular, it does exceedingly well in the event that a higher percentage of the roles in the system are defined using intersection containment, as it can rely more on forward chaining.

5.4 Future Optimizations

While the use of dynamic views described in this paper has the advantage of seamlessly adapting to changes in underlying policy, the previous experiments show that this comes at a significant cost as the policies become increasingly nested. One solution to this problem is to compile logical policies into *materialized*, rather than dynamic, views. At a high level, a materialized view is generated by issuing the query that defines the view and saving the result set as a table within the database. Further, a materialized view can be indexed, while a dynamic view cannot. The result is that queries to any role managed within the database could be executed at the same speed as queries to the base tables. The data shown in Figures 5(a)–(b) suggest that this would imply large performance gains. Although MySQL currently does not support materialized views, we plan to explore this in the future using a DBMS like PostgreSQL or Oracle.

The efficiency gains that can be realized by materialized views do not come without cost, however. In particular, materialized views (i) require much more space within the database (1000s of rows) when compared to dynamic views (stored as a single query), and (ii) must be regenerated on-the-fly to remain consistent with changes made to the tables and views over which they are defined (e.g., when logical policies are edited). Fortunately, we do not see either of these issues causing a problem in practice. With respect to (i), disk and memory space are abundant in even low-end enterprise servers. Similarly, Figures 5(a)-(b) imply that (ii) can be addressed, as the cost of carrying out even complete updates to a materialized view would be very low. Often times, complete updates are not even needed to keep materialized views consistent, so Figures 5(a)-(b) could be viewed as a conservative estimate of update cost within such a system.

6. RELATED WORK

Digital credentials are one of the main techniques for access control in cross-domain collaboration and resource sharing. Due the decentralized nature of such applications, digital credentials are largely extended from simple bindings between public keys and identities to signed statements with rich semantics, including roles, properties, logical inference rules and even arbitrary programs. Extensive work has been done in the areas of trust management [2, 3, 5, 20], trust negotiation [19, 27, 29, 30], and distributed proofs [1, 12, 15, 24]. These systems consider both centralized credentials (where credentials are maintained in a well-known repository) and decentralized credentials (where credentials are distributed among multiple entities). In terms of trust evaluation, a majority of work in the above areas focuses on compliance checking, i.e., determining through logical inferences

Parameter	Description	Values
c	Number of companies or VOs at each level	3
u	Number of users in each base company	100, 500, 1000
r	The number of roles in each company and VO	30
rpu	The number of roles assigned to each user	10
nm	The number of companies or VOs making up each VO	3
p_i	Probability of generating an intersection containment credential	1.0, 0.9, 0.7
p_l	Probability of generating a linking containment credential	0.0, 0.05, 0.15
p_a	Probability of generating an aggregate containment credential	0.0, 0.05, 0.15
mi	Maximum number of roles comprising an intersection containment	3
mc	Maximum number of credentials defining each role	3

Table 1: System parameters for the virtual organization scenario.

whether a principal belongs to a particular role or possesses certain privileges. Except for [11], no previous work considered to build trust management systems on top of commodity DBMSs. As mentioned before, the work in [11] focuses on using databases as credential stores for trust management, while the focus of this paper is to utilize the data processing capability of DBMSs to improve trust policy evaluation performance.

Though reputation-based trust has been studied extensively in in the context of agent systems [13], online auctions, pervasive computing and P2P systems [14,28], its integration with credential-based trust is only considered recently [4,7]. As mentioned above, Lee and Yu [18] proposed CTM, a formal framework to represent and interpret the combination of vertical and horizontal trust. They do not consider how to evaluate trust policies in CTM.

Traditional access control is often identity-based. Most is implemented through access control lists (ACL), capability lists or access control trips. Due to the simplicity of such access control policies, the checking for a particular privilege of a principal is straightforward and quite efficient. The efficiency of policy evaluation becomes important when access control policies become complicated. One major effort is CPOL [8], which achieves high-perform policy evaluation through indexing and caching techniques. Though the policy language in CPOL is relatively expressive, it focuses on simple delegation and location and temporal constraints. CTM on the other hands supports more types of delegations as well as the combination of credentials with reputations. Therefore, the techniques in CPOL cannot be directly applied to our problem.

Li et al. [22] designed credential chain discovery algorithms to answer the three types of queries for *RT*. Their algorithms do not take advantage of supports from any existing software systems or tools. In this paper, we try to utilize as much functionality as possible for DBMSs to make the trust evaluation engine efficient, lightweight, and easy to deploy.

As CTM is based on *RT*, which is based on logical inferences, it is also possible to build a trust policy evaluation engine using logic programming languages such as Prolog and Datalog. However, such languages are often not well supported in commercial information systems. In this paper, we emphasize on utilizing the data processing capabilities in existing IT infrastructures to make trust evaluation engines not only efficient but also lightweight and easy to deploy.

7. CONCLUSION

Despite much promising research into trust management approaches to authorization, relatively little attention has been given to exactly how these technologies can be effectively deployed. In this paper, we addressed one potential avenue for supporting the deployment of logical trust management approaches by means of leveraging a key fixture of the IT landscape: the relational database. To this end, we developed a two-tiered framework comprised of a logical front end and a relational back end. We showed that simple role membership credentials and reputation feedback reports can be efficiently managed in the DBMSs base tables, and we presented a correct and complete method for compiling collections of logical CTM credentials into dynamic views stored in the database. This relational representation can be used to quickly perform proof of compliance checks or enumerate all members of a given role. We also developed a novel algorithm for answering capability queries that takes advantage of forward chaining across the system's logical ruleset whenever possible, while also leveraging the database's ability to quickly perform complex joins and aggregations in situations where forward chaining would be inefficient. An initial evaluation of an unoptimized prototype implementation shows that our proposed approach is a viable solution in even when making pessimistic assumptions regarding credential complexity.

In the future, we plan to move beyond the research prototype phase and optimize the relational side of our framework. In Section 5.4 we discussed how the use of materialized, rather than dynamic, views would likely lead to immense performance gains for our system. We plan to examine the relative costs involved with using materialized views—including increased disk and memory utilization, as well as overheads associated with keeping these views consistent with changes to the underlying ruleset—and carry out a detailed cost/benefit analysis. We also plan to investigate effective means applying our proposed framework within decentralized organizations. In particular, we hope to leverage information gleaned from centralized query plans to develop efficient algorithms for processing decentralized trust queries across multiple domains. Another interesting problem is to investigate how to support circular dependent roles in DBMSs through, for example, recursive queries.

Acknowledgments. The research described in this paper was supported by the National Science Foundation under awards CCF-0916015 and CCF-0914946.

8. REFERENCES

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

[2] M. Y. Becker, C. Fournet, and A. D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 2009.

[3] M. Y. Becker and P. Sewell. Cassandra: Distributed access control policies with tunable expressiveness. In *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 159–168, June 2004.

[4] B. K. Bhargava and Y. Zhong. Authorization based on evidence and trust. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 94–103, Aix-en-Provence, France, Sept. 2002.

[5] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the IEEE Conference on Security and Privacy*, pages 164–173, May 1996.

[6] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance checking in the PolicyMaker trust management system. In *Proceedings of the Second International Conference on Financial Cryptography*, number 1465 in Lecture Notes in Computer Science, pages 254–274. Springer, Feb. 1998.

[7] P. Bonatti, C. Duma, D. Olmedilla, and N. Shahmehri. An integration of reputation-based and policy-based trust management. In *Sematic Web and Policy Workshop*, Galway, Ireland, Nov. 2005.

[8] K. Borders, X. Zhao, and A. Prakash. CPOL: High-performance policy evaluation. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS)*, pages 147–157, Nov. 2005.

[9] B. Carminati, E. Ferrari, and A. Perego. Enforcing access control in web-based social networks. *ACM Transactions in Information and System Security*. to appear.

[10] B. Carminati, E. Ferrari, and A. Perego. A decentralized security framework for web-based social networks. *International Journal of Information Security and Privacy*, 2(4):22–53, 2008.

[11] S. D. C. di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Trust management services in relational databases. In *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 149–160, Mar. 2007.

[12] T. Jim. SD3: A trust management system with certified evaluation. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 106–115, May 2001.

[13] A. Josang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 43(2):618–644, 2007.

[14] S. Kamvar, M. Schlosser, and H. Garcia-Molina. EigenRep: Reputation Management in P2P Networks. In *Twelfth International World Wide Web Conference*, 2003.

[15] A. J. Lee, K. Minami, and N. Borisov. Confidentiality-preserving distributed proofs of conjunctive queries. In *ACM Symposium on Information, Computer, and Communication Security (ASIACCS)*, Mar. 2009.

[16] A. J. Lee and M. Winslett. Enforcing safety and consistency constraints in policy-based authorization systems. *ACM Transactions on Information and System Security*, 12(2), Dec. 2008.

[17] A. J. Lee and M. Winslett. Towards an efficient and language-agnostic compliance checker for trust negotiation systems. In *Proceedings of the 3rd ACM Symposium on Information, Computer and Communications Security (ASIACCS 2008)*, pages 228–239, Mar. 2008.

[18] A. J. Lee and T. Yu. Towards a dynamic and composite model of trust. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 217–226, June 2009.

[19] J. Li, N. Li, and W. H. Winsborough. Automated trust negotiation using cryptographic credentials. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 46–57, Nov. 2005.

[20] N. Li and J. C. Mitchell. RT: A role-based trust-management framework. In *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, pages 201–212, Apr. 2003.

[21] N. Li, J. C. Mitchell, and W. H. Winsborough. Beyond proof-of-compliance: security analysis in trust management. *Journal of the ACM*, 52(3):474–514, 2005.

[22] N. Li, W. Winsborough, and J. Mitchell. Distributed Credential Chain Discovery in Trust Management. *Journal of Computer Security*, 11(1), Feb. 2003.

[23] N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, 2003.

[24] K. Minami and D. Kotz. Secure context-sensitive authorization. *Journal of Pervasive and Mobile Computing (PMC)*, 1(1):123–156, Mar. 2005.

[25] I. Molloy, N. Li, T. Li, Z. Mao, Q. Wang, and J. Lobo. Evaluating role mining algorithms. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 95–104, 2009.

[26] J. Vaidya, V. Atluri, and J. Warner. Roleminer: mining roles using subset enumeration. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)*, pages 144–153, 2006.

[27] W. H. Winsborough, K. E. Seamons, and V. E. Jones. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition*, Jan. 2000.

[28] L. Xiong and L. Liu. A reputation based trust model for peer-to-peer ecommerce communities. In *IEEE International Conference on E-Commerce (CEC)*, 2003.

[29] T. Yu, M. Winslett, and K. E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies in automated trust negotiation. *ACM Transaction on Information and System Security (TISSEC)*, 6(1):1–42, Feb. 2003.

[30] C. C. Zhang and M. Winslett. Distributed authorization by multiparty trust negotiation. In *ESORICS 2008*, pages 282–299, Oct. 2008.