RT: A Role-based Trust-management Framework[*]

Ninghui Li John C. Mitchell
Department of Computer Science, Stanford University
Gates 4B, Stanford, CA 94305-9045
{ninghui.li, mitchell}@cs.stanford.edu

Abstract

The RT Role-based Trust-management framework provides policy language, semantics, deduction engine, and pragmatic features such as application domain specification documents that help distributed users maintain consistent use of policy terms. This paper provides a general overview of the framework, combining some aspects described in previous publications with recent improvements and explanation of motivating applications.

1 Introduction

The Agile Management of Dynamic Collaboration (AMDC) project develops infrastructure and support for secure, trusted dynamic coalitions, with emphasis on the following areas:

1. *Trust management*: Support for distributed management of trust relations, in a form suitable for determining the degree of trust associated with a potential peer, service or coalition.
2. *Search and selection of coalitions and partners*: Protocols and facilities for authenticated selection and communication with partners and coalitions.
3. *Mobile code risks*: Security for dynamic installation of executable code, as required to support dynamic system self-configuration.

The focus of this paper is on RT, a Role-based Trust-management framework [20, 19, 18] developed in the course of the AMDC project. This work supports approaches to the other two areas in the project, as outlined in Section 8. RT aims to address access control and authorization problems in large-scale, decentralized systems. Such problems arise, for example, when independent organizations enter into coalitions whose membership and very existence change rapidly. A coalition may be formed by several autonomous organizations wishing to share resources. While sharing resources, each organization retains ultimate authority over the resources it controlled prior to entering the coalition. We call such systems *multicentric collaborative systems*, since they have no single central authority.

RT combines the strengths of Role-Based Access Control (RBAC) [22] and trust-management (TM) systems. From RBAC, it takes the notions of role, interposed in the assignment of permissions to users to aid organizing those assignments, and of sessions and selective role activations. From TM, RT takes principles of managing distributed authority through the use of credentials, as well as some clear notation denoting relationships between those authorities, e.g., localized name spaces and linked local names from SDSI (Simple Distributed Security Infrastructure) [21, 11]. From Delegation Logic [17], RT takes the logic-programming-based approach.

In addition, the RT framework makes the following contributions.

- RT has policy concepts such as intersections of roles, role-product operators, manifold roles, and delegation of role activations. These concepts can express policies that are not possible to express in existing systems; they can also express some other policies in more succinct or intuitive ways.
- Most logic-based TM systems use DATALOG as the semantic foundation. In RT, we use the more expressive Constraint DATALOG, which enables one to express permissions regarding structured resources, while at the same time preserving the desirable properties of DATALOG.
- RT supports credential chain discovery when credential storage is distributed, through a goal-directed chain discovery algorithm. RT also has a storage typing system which guarantees that distributed credential chains can be discovered and guides efficient discovery.
- RT addresses the issue of vocabulary agreement, using Application Domain Specification Documents (ADSDs) which enables RT to have strongly typed credentials.

[*]Supported in part by DARPA through SPAWAR contract N66001-00-C-8015.

The design of RT is driven by the requirements of practical applications. In order to better understand practical issues associated with sharing information among different parties, we have designed and implemented two demonstration applications, Digital U-STOR-IT, a secure web-based file sharing application, and August, a distributed scheduling application. Through ongoing development of specific policies for these applications, we have identified the need for specific policy features.

Some parts of RT are the result of collaboration with the Attribute-Based Access Control (ABAC) project at Network Associates Laboratories. RT is used in the ABAC project as the underlying policy language [25, 24].

The rest of this paper is organized as follows. We give an overview of TM and discuss related work in Section 2. We then give an overview of RT in Section 3. In Sections 4, 5, 6, and 7, we discuss the language, the semantic foundation, chain discovery issues, and the implementation as well as applications of RT. In Section 8, we briefly discuss TM-related work on service selection and mobile code that have been conducted in the AMDC project. We conclude in Section 9.

2 Overview of Trust Management

Traditional access control mechanisms make authorization decisions based on the identity of the requester. However, in decentralized or multicentric environments, the resource owner and the requester often are unknown to one another, and access control based on identity may be ineffective. Trust management is an approach to distributed access control and authorization, in which access control decisions are based on *policy statements* made by multiple principals.

2.1 An Abstract TM Framework

We present an abstract framework for trust-management systems, which consists of three aspects: language, deduction, and infrastructure.

Language

A TM language has a mechanism for identifying principals, a syntax for specifying policy statements and queries, and a semantic relation that determines whether a query is true given a set of policy statements.

Principals can issue policy statements, make requests, and be authorized to perform actions. Policy statements describe the properties of principals and how to derive one property from other properties. Example properties include membership in a group, being a student, having a birthday that is a certain date, membership in a role within an organization, receiving delegation of a permission or role, etc. In decentralized environments, authenticity and integrity of policy statements need to be protected. In many TM systems, principals are identified with public keys, and non-local policy statements are digitally signed. Signed policy statements are called *credentials*.

When a query corresponds to an access request, the semantic relation of a TM language defines a notion commonly known as *proof-of-compliance*: "Does a set of policy statements prove that a request should be authorized?" It is also helpful for a TM language to support more advanced queries such as finding out all principals that are authorized to access a resource, finding out what access permissions a specific principal has, etc.

Deduction

Deduction implements the semantic of the language. A TM system may have the following deduction engines (algorithms). A *proof checking engine* takes a set of policy statements, a query, and an answer as input, and verifies that the answer is true. The answer may be equipped with proofs (or proof hints) to make proof checking simpler. A *proof construction engine* (also known as a *chain discovery engine*) takes a set of policy statements and a query as input, and finds an answer, optionally constructing proofs (or proof hints). In systems that may have a large number of (e.g., millions of) policy statements stored in a decentralized manner, a chain discovery engine does not have the complete set of policy statements as input, and should be able to start evaluation with a query and an incomplete set of policy statements and to interleave retrieving policy statements and inferencing.

Infrastructure

The infrastructure aspect of a TM system includes support for policy statement creation, storage, distribution, revocation, etc. These issues are less coupled with language and deduction. They are similar to those in public key infrastructure and have been extensively studied.

2.2 Usage Scenarios of TM

We now describe several scenarios in which TM systems could be useful. These are for illustration purposes and are not intended to be exhaustive or a rigid classification.

More flexible control in centralized environments

Trust-management systems can be used to achieve more flexible control in centralized environments such as operating systems, web-based file sharing systems, etc. In UNIX, the owner of a file can grant the permission to access a file to the group that the owner is in. However, when an ordinary user wants to share a file with, say, a group of collaborators working on a paper, the user cannot do this without the involvement of an administrator. It would be useful for each user to define groups according to their own needs and assign permissions to access a file more flexibly. In some applications, it would also be useful to delegate the ability to administrate a file's permission to other users.

Multicentric collaboration systems

In multicentric collaboration systems, users belonging to one organization need to access resources controlled by other organizations in a coalition. Trust management allows resource controllers to delegate certain forms of policy setting to users in other organizations. This may be done in a controlled manner, without delegating arbitrary access control. For example, access decisions may be based on both the resource controllers' local policy statements that encode the relationship between the two collaborating organizations and the credentials that encode the requesters' roles/positions in their organizations.

Loosely-coupled decentralized systems

A TM system can provide a more expressive global-scale public-key infrastructure, by having digital credentials for driver licenses, student IDs, credit cards, organization memberships, trusting relationships regarding these digital credentials, and so on. In these cases, online transaction may require the combination of these credentials.

Distributed computing environments

In addition to access requests from individuals, trust management can also be used to control access by processes within a distributed computing environment. Suppose a user starts a session, activating some of his eligible roles, and then issues a request. To fulfill this request, the session process starts a second process on behalf of the user, which invokes a third process, which is running on a different host, so as to access back-end services needed to complete the requested task. Each of these processes must be delegated the authority to act on the user's behalf, and the first two must pass that authority to the processes they initiate.

2.3 Related Work

The term "trust management" was coined in [6], in which the PolicyMaker system was introduced. The second generation of PolicyMaker is KeyNote [5]. KeyNote and version 1 of Simple Public Key Infrastructure (SPKI) can be called *permission-based TM systems*, since they use policy statements only to delegate permissions. Each statement delegates certain permissions from its issuer to its subject. A chain of one or more statements acts as a capability, granting certain permissions to the subject of the last statement in the chain.[1] However, in these permission-based systems, one cannot express the fact that the issuer grants permissions to all principals that have a certain property. As a result, the delegation relationships in such system are quite limited. See [19, 17] for more detailed discussions of this limitation.

The ABLP Logic [1, 16] is a propositional modal logic designed mainly for authentication in distributed systems. The logic can be used to determine, e.g., in a tightly coupled distributed systems, who originally made a request that has gone through multiple encrypted channels and processes on multiple hosts. The core concept in ABLP logic is a "speaks for" relation among principals; that A speaks for B means that, if principal A makes a statement, then we can believe that principal B makes it, too. It was pointed out in [2] that the modal operators in ABLP Logic can be defined using a general higher-order propositional logic. In the framework described in [2], specific systems can define their own operators, clients are responsible to construct the proof that they can access certain resources, and the server only needs a generic verifier for the general logic to verify the proof. A system based on this concept is implemented in [4]. The logics in [1, 16, 2] are based on propositional logic; they cannot describe permissions about structured resources. In these logics, a principal cannot delegate e.g., the permission to access all directories and files under a certain directory, the authority to issue student certificates but only when the name of the school is a certain value, or the permission to approve purchase orders whose value is below a threshold.

Among previous TM systems, the ones closest to RT are Delegation Logic [17] and SPKI/SDSI [11]. The four new contributions of RT have been listed in Section 1 and will be elaborated in the coming sections.

3 An Overview of the RT framework

In this section, we give an overview of RT.

Principals and Roles

In RT, principals can be uniquely identified individuals, processes, public keys, etc.

A central organizing concept in RT is the notion of *roles*. Each RT principal has its own name space for roles, similar to localized name spaces in SDSI. A role is named by a principal and a *role term*. For example, if K_A is a principal and R is a role term, then $K_A.R$ is the role R defined by principal K_A and can be read as K_A's R role. A role term consists of a role name and zero or more parameters. Only K_A can issue policy statements defining the role $K_A.R$; these statements determine members of $K_A.R$. For example, K_A can define $K_A.R$ to include another principal K_B's role, effectively delegating some control over the role $K_A.R$ to K_B. When K_A issues multiple statements defining a role, the role contains the union of all the resulting principals.

[1]Because KeyNote and SPKI have thresholds, a capability could be a directed graph of credentials. This does not affect our discussion of their limitations below.

RT has *single-element roles* and *manifold roles*. The semantics of a single-element role is a set of principals. The notion of single-element roles unifies several concepts in access control and trust-management literature, including groups in many systems, identity in identity certification systems such as X.509, roles and permissions in RBAC, names in SDSI, authorization tags in SPKI [11], and attributes in attribute certificates. It is possible to unify these concepts because the common mathematical underpinning of the semantics of these concepts is a set of principals. A group is clearly a set of principals. An identity is a set of principals corresponding to one physical user; some systems require the set to contain just one principal. A role in RBAC can be viewed as a set of principals who are members of this role. The role hierarchy relationship that $K_B.R$ is more powerful than $K_A.R$ can be viewed a definition that all members of $K_B.R$ are also members of $K_A.R$. A permission corresponds to a set of principals who have the permission. Granting a permission to a principal amounts to making the principal a member of the set corresponding to the permission. Granting a permission to a role amounts to asserting that the set corresponding to the permission includes as a subset the set corresponding to the role. A name in SDSI is also resolved to a set of principals. An attribute may be identified with the set of principals who have the attribute.

The notion of manifold roles generalizes that of single-element roles to allow each member of a role to be a principal set, instead of a principal. That the principal set $\{K_1, K_2\}$ is a member of the manifold role $K_A.R$ means that K_1 and K_2 together have the privileges associated with $K_A.R$, but neither of them acting alone has that privilege. The semantics of a manifold role is a set of principal sets. Manifold roles are introduced to support separation-of-duty (SoD) policies in a more expressive way than threshold structures.

In many scenarios, a user prefers not to exercise all his rights. An administrator often logs in as an ordinary user to perform ordinary tasks. In another example, a user is temporarily delegated certain access rights by his manager during his manager's absence. The user will often want to exercise only his customary rights, wishing to use his temporary rights only when explicitly working on his manager's behalf. This notion is related to the least privilege principle and is supported by many systems. In RBAC, it is supported by the notion of sessions. A user can selectively activate some of his eligible roles in a session. This can be viewed as a delegation of role activations from the user to the session. A natural generalization of user-to-session delegation of role activations is process-to-process delegation of those role activations. The need for this is particularly acute in distributed computing environments. The RT framework has delegation of role activations, which can be used to express selective role activations, delegation of role activations, and access requests supported by a subset of the requesting principal's roles.

Semantic foundation

In [19], DATALOG was used as the semantic foundation for the RT framework. Policy statements in RT are translated into DATALOG rules. This guarantees that the semantics is precise, monotonic, and algorithmically tractable. In [18], Constraint DATALOG was introduced as a more expressive foundation for TM languages, while preserving the advantages of DATALOG. The requirement that RT policy statements can be translated into rules in DATALOG with tractable constraints is the main design constraint on expressivity in the design of the RT framework.

Support for distributed chain discovery

Most previous work addressing the credential chain discovery problem assumes that one has already gathered all the potentially relevant credentials in one place and does not consider how to gather these credentials. The assumption that all credentials are stored in one place does not hold in many applications that use trust management for decentralized control. In these systems, credentials are issued and stored in a distributed manner.

Distributed discovery requires an evaluation procedure that is goal-directed, expending effort only on chains that involve the requester and the access mediator, or its trusted authorities, and considering credentials in a demand-driven manner. Goal-directed algorithms can be contrasted with bottom-up algorithms, which require collecting all credentials before commencing evaluation. In the Internet, with distributed storage of millions of credentials, most of them unrelated to one another, goal-directed techniques will be crucial. Distributed credential chain discovery also requires a procedure that can begin evaluation with an incomplete set of credentials, then suspend evaluation, issue a request for credentials that could extend partial chains, then resume evaluation when additional credentials are obtained, and iterate these steps as needed.

In [20], goal-directed credential chain discovery algorithms for RT_0, the basic component of RT, were presented. These algorithms work when credentials are stored in a distributed manner. Issues related to distributed credential storage are also addressed.

Support for vocabulary agreement

One distinguishing feature of the RT framework is that it directly addresses the issue of vocabulary agreement. When credential chains delegate access permissions of resources, all the principals involved in the chain need to use consistent terminology to specify resource permissions and delegation conditions. When different credential issuers use incompatible schemes, their credentials cannot be meaningfully combined. Some intended permissions may not be granted, or, when schemes intended for different purposes accidentally interact, unintended authorization may follow. Some systems do not address this issue at all; others try to come up with one vocabulary for all applications. Our philosophy is that, although different applications often share common policy concepts, they need to be able to use different vocabularies. In RT, we address this issue through a scheme inspired by XML namespaces [7].

We introduce *application domain specification documents (ADSDs)*. Each ADSD is globally uniquely identified. One way to uniquely identify an ADSD is to use an URI pointing to the document together with a collision-free hash of the document. An ADSD declares a suite of related data types and role names, called a *vocabulary*. Credentials, when using a role name, refer to the ADSD in which the role name is declared. This enables RT to have strongly typed credentials and policies. This feature helps ensure interoperability and reduce the possibility of errors in writing policies and credentials and unintended interaction of credentials. One can think of ADSDs as `.h` files in C programs and credentials as `.c` files. Data types and role names are declared in ADSDs, and credentials must use these role names in a type-consistent way.

The notion of vocabularies is complimentary to the notion of localized name spaces for roles. Each addresses a distinct name space issue. For example, an accrediting board might issue an ADSD that declares the role name "student". This defines the names and data types of the role's parameters. Such parameters may include university name, student name, program enrolled in, etc. The ADSD may also contain description of the conditions under which a principal should be made a member of the student role, e.g., it may require a principal be registered in a degree program. Then a university StateU can use this ADSD to issue credentials defining StateU.student. Although using a vocabulary created by another principal, StateU is still the authority over who is a member of the role StateU.student.

4 The RT family of TM languages

The most basic language in the RT family of TM languages is RT_0, which was presented in [20]. In RT_0, role terms are simply role names and do not take any parameters. In [19], several additional components of RT were introduced. RT_1 adds to RT_0 parameterized roles. RT^T provides manifold roles and role-product operators, which can express threshold and separation-of-duty policies. RT^D provides delegation of role activations, which can express selective use of capacities and delegation of these capacities. RT^T and RT^D can be used, together or independently with RT_0 or RT_1.

4.1 RT_0: Defining Roles

In RT_0, policy statements take the form of *role definitions*. A role definition has a head and a body. The *head* of a role definition has the form $K_A.R$, in which K_A is a principal, R is a role term, which is simply a role name in RT_0.

Simple Member $K_A.R \longleftarrow K_D$

The body consists of one principal K_D. This defines the principal K_D to be the member of the role $K_A.R$.

Simple Containment $K_A.R \longleftarrow K_B.R_1$

The body consists of one role. This defines the role $K_A.R$ to contain (every principal that is a member of) the role $K_B.R_1$.

Linking Containment $K_A.R \longleftarrow K_A.R_1.R_2$

We call $K_A.R_1.R_2$ a *linked role*. This defines $K_A.R$ to contain $K_B.R_2$ for every K_B that is a member of the role $K_A.R_1$. Note that the body also starts with K_A and has only two role terms, this limitation does not affect the expressive power, as one can use intermediate roles and additional statements to express long linked roles. See [20] for the rationale of this limitation.

Intersection Containment

$$K_A.R \longleftarrow K_{B_1}.R_1 \cap \cdots \cap K_{B_k}.R_k$$

This defines $K_A.R$ to contain the intersection of all the roles $K_{B_1}.R_1, \cdots, K_{B_k}.R_k$.

Simple Delegation $K_A.R \Longleftarrow K_B : K_C.R_2$

The part after the colon (i.e., $K_C.R_2$) is optional. This means that K_A delegates its authority over R to K_B. In other words, K_A trusts K_B's judgement on assigning members to R. When $K_C.R_2$ is present, K_A wants to control its delegation such that K_B can only assign members of $K_C.R_2$ to be members of $K_A.R$. In RT_0, this can be viewed as a convenient shorthand for "$K_A.R \longleftarrow K_B.R \cap K_C.R_2$".

Linking Delegation $K_A.R \Longleftarrow K_A.R_1 : K_C.R_2$

The part after the colon (i.e., $K_C.R_2$) is optional. This means that K_A delegates its authority over R to members of $K_A.R_1$, and the delegation is restricted to members of $K_C.R_2$. This implies $K_A.R \longleftarrow K_A.R_1.R \cap K_C.R_2$.

Simple delegation and linked delegation are definable using the other four forms of definitions. (Intermediate roles are needed for linking delegation.) These two forms of delegation become necessary when roles can take parameters and can be extended to have additional parameters.

Example 1 A fictitious Web publishing service, EPub, offers a discount to anyone who is both a preferred customer of its parent organization, EOrg, and an IEEE member. EOrg considers university students to be preferred customers. EOrg delegates the authority over the identification of students to principals that are accredited universities. To identify such universities, EOrg accepts accrediting credentials issued by the fictitious Accrediting Board for Universities (ABU). The following credentials prove that Alice is eligible for the discount:

EPub.discount $\longleftarrow$ EOrg.preferred $\cap$ IEEE.member
EOrg.preferred $\longleftarrow$ EOrg.university.student
EOrg.university $\longleftarrow$ ABU.accredited
ABU.accredited $\longleftarrow$ StateU
StateU.student $\longleftarrow$ Alice
IEEE.member $\longleftarrow$ Alice

4.2 RT_1: Adding Parameters

RT_1 adds to RT_0 parameterized roles. Role definitions in RT_1 have the same format as those in RT_0. In RT_1, a role term takes the form $r(p_1, \ldots, p_n)$, in which r is a role name, and each p_j takes one of the following three forms: $name = c$, $name = ?X[\in S]$ (the $\in S$ part is optional), and $name \in S$, where $name$ is the name of a parameter of r, c is a constant of the appropriate type, $?X$ is a variable, and S is a *value set* of the appropriate type. Variables are used to make two parameters in one role definition equal. A value set can be viewed as a (typically constraint-based) representation of a set of values.

Example 2 A firewall administrator uses K_{FW}, the key of the firewall to issue a credential that gives a system administrator, SA, the authority to grant to anyone who has a Stanford ID the permission to connect to any host in the domain "stanford.edu". Later, SA grants to Alice the permission to connect to the host "cs.stanford.edu" at any port between 8000 and 8443.

K_{FW}.perm(host$\in$ descendants('stanford.edu'))
  $\Longleftarrow K_{\mathrm{SA}} : K_{\mathrm{Stanford}}$.stanfordID()
K_{SA}.perm(host = 'cs.stanford.edu', port$\in$ [8000..8443])
  $\longleftarrow K_{\mathrm{Alice}}$

If Alice has a Stanford ID, then when she requests to connect to the host "cs.stanford.edu" at port 8443, FW should allow this connection.

In these credentials, "descendants" is special value set constructor for tree domains (see Section 5). In the first credential, the port parameter does not appear in the role term K_{FW}.perm(host$\in \cdots$); this means that the port parameter is not constrained in this delegation. Similarly, the role term K_{Stanford}.stanfordID() in the first credential has no parameters, since no constraint is required.

4.3 RT^T: Supporting Separation of Duty

The separation-of-duty (SoD) security principle [9, 23] requires that two or more different persons together be responsible for the completion of a sensitive task. SoD can be used to discourage fraud by requiring collusion among principals to commit fraud.

Both SPKI and KeyNote allow delegation to k-out-of-n threshold structures, in which one explicitly lists the n principals. It has been argued before that such threshold structures are inconvenient. For example, to express a policy that requires two different cashiers, a SPKI or KeyNote policy statement needs to explicitly list all the cashiers, and this statement needs to be changed each time members in the cashier role change. Delegation Logic has the more expressive dynamic threshold structures, which are satisfied by the agreement of k out of a set of principals that satisfy a specified condition.

In RBAC, SoD is often achieved by using constraints such as mutual exclusion among roles [22, 23] and requiring cooperation of mutually exclusive roles to complete sensitive tasks. Because no principal is allowed to simultaneously occupy two mutually exclusive roles, sensitive tasks can be completed only by cooperation of principals.

Threshold structures require agreement of different principals drawn from a single set. Mutually exclusive roles require agreement among several disjoint sets (one out of each set). Each of the two mechanisms has limitation, as it cannot achieve what the other mechanism achieves.

RT^T introduces the notion of *manifold roles* to achieve both agreement of multiple principals from one set and from disjoint sets. Similar to a role, which defines a set of principals, a manifold role defines a set of *principal sets*, each of which is a set of principals whose cooperation satisfies the manifold role. Manifold roles are defined by role expressions constructed using either of the two *role-product operators*: $\odot$ and $\otimes$.

Product Containment

$$K_A.R \longleftarrow K_{B_1}.R_1 \odot \cdots \odot K_{B_k}.R_k$$

This defines the role $K_A.R$ to contain every principal set p such that $p = p_1 \cup \cdots \cup p_k$ and for each $1 \leq j \leq k$, p_j is a member of $K_{B_j}.R_j$.

Exclusive Product Containment

$$K_A.R \longleftarrow K_{B_1}.R_1 \otimes \cdots \otimes K_{B_k}.R_k$$

This defines the role $K_A.R$ to contain every principal set p that satisfies the following condition: $p = p_1 \cup \cdots \cup p_k$, $p_i \cap p_j = \emptyset$ for $1 \leq i \neq j \leq k$, (p_i and p_j are non-intersecting), and p_j is a member of $K_{B_j}.R_j$ for each $1 \leq j \leq k$.

Example 3 A bank FB has three roles: manager, cashier, and auditor. FB's policy requires that a certain transaction be approved by a manager, two cashiers, and an auditor. The two cashiers must be different. A manager who is also a cashier can serve as one of the two cashiers. And the auditor must be different from the other parties in the transaction.

FB.twoCashiers $\longleftarrow$ FB.cashier $\otimes$ FB.cashier
FB.mgrCashiers $\longleftarrow$ FB.manager $\odot$ FB.twoCashiers
FB.approval $\longleftarrow$ FB.auditor $\otimes$ FB.mgrCashiers

4.4 RT^D: Delegation of Role Activations

We introduce RT^D to handle delegation of the capacity to exercise role memberships. RT^D adds the notion of delegation of role activations to the RT framework. That a principal D activates the role $A.R$ to use in a session B_0 can be represented by a *delegation credential* issued by D, "$D \xrightarrow{D \text{ as } A.R} B_0$". We call "$D$ as $A.R$" a role activation. B_0 can further delegate this role activation to B_1 by issuing the credential, "$B_0 \xrightarrow{D \text{ as } A.R} B_1$". A principal can issue multiple delegation credentials to another principal. Also, several role activations can be delegated in one delegation credential. This is viewed as a shorthand for multiple delegation credentials.

A delegation credential can also contains a keyword "all". For example, "$B_0 \xrightarrow{\texttt{all}} B_1$" means that B_0 is delegating all role activations it has to B_1; and "$B_0 \xrightarrow{D \text{ as } \texttt{all}} B_1$" means that B_0 is delegating to B_1 those of B_0's role activations in which D is activating the roles.

A request in RT^D is represented by a delegation credential that delegates from the requester to the request. For example, that B_1 requests to read fileA in the capacity of "D as $A.R$" can be represented by: $B_1 \xrightarrow{D \text{ as } A.R}$ fileAccess(read, fileA). This request should be authorized if D is a member of the role $A.R$, the role $A.R$ has read access to fileA, and there is a chain of delegation from D to B_1 about the role activation $A.R$. Note that fileAccess(read, fileA) is not a principal. This delegation should be interpreted as being from B_1 to a dummy principal representing the request fileAccess(read, fileA). An RT^D system assigns a unique dummy principal to each request.

That B_1 is making the request *req* using all its capacities is represented by $B_1 \xrightarrow{\texttt{all}} req$. Delegation of role activations is delegation of the capacity to act in a role. It is a different kind of delegation from delegation of authority to define a role, as in a role-definition credential "$A.R \longleftarrow B.R$".

4.5 Examples Using RT_1^{DT}

Example 4 In an organization SOrg, any purchasing order has to be submitted and approved before it is placed. Any employee can submit a purchasing order. A manager can approve an order. A manager is also an employee; however, a manager cannot approve his own order. This can be represented as follows:

SOrg.place $\longleftarrow$ SOrg.submit $\otimes$ SOrg.approve
SOrg.submit $\longleftarrow$ SOrg.employee
SOrg.approve $\longleftarrow$ SOrg.manager
SOrg.employee $\longleftarrow$ SOrg.manager

Suppose that both Alice and Bob and managers:

SOrg.manager $\longleftarrow$ Alice
SOrg.manager $\longleftarrow$ Bob

Alice can submit an order by issuing:

Alice $\xrightarrow{\texttt{Alice as SOrg.employee}}$ order(orderID)

And Bob can approve it by issuing:

Bob $\xrightarrow{\texttt{Bob as SOrg.approve}}$ order(orderID)

Then the order should be approved.

The scenario described in the following example is originally from [1].

Example 5 A server S authorizes fileA to be deleted if it is requested from a good workstation on behalf of a user. S knows that alice is a user and trusts CA in certifying public keys for users. S knows that ws1 is a good workstation and trusts CA in certifying public keys for workstations. These are expressed in the following credentials:

S.del(fileA) $\longleftarrow$ S.user $\otimes$ S.goodWS
S.user $\longleftarrow$ K_{CA}.userCert(alice)
S.goodWS $\longleftarrow$ K_{CA}.machineCert(ws1)

The following are credentials issued by CA:

K_{CA}.userCert(alice) $\longleftarrow$ K_alice
K_{CA}.machineCert(ws1) $\longleftarrow$ K_ws1

A work station stores its private key in tamper-resistant firmware. When it boots, it generates a key pair for the operating system and issues a credential to delegate the activation of S.goodWS to the new key. When the user alice logs into a workstation ws1, a new process p1 is set up and a new key pair is generated. Through p1, alice then makes a request to the server S to delete fileA. The process p1 sets up a secure channel Ch to the server, and then sends the request through the channel. The following are delegation credentials that are needed.

K_ws1 $\xrightarrow{\texttt{K\_ws1 as S.goodWS}}$ K_os1
K_os1 $\xrightarrow{\texttt{K\_ws1 as S.goodWS}}$ K_p1
K_alice $\xrightarrow{\texttt{K\_alice as S.user}}$ K_p1
K_p1 $\xrightarrow{\texttt{K\_ws1 as S.goodWS, K\_alice as S.user}}$ K_Ch

The request sent by K_Ch to delete fileA on behalf of user alice working on a good workstation is represented as:

K_Ch $\xrightarrow{\texttt{K\_ws1 as S.goodWS, K\_alice as S.user}}$ del(fileA).

And this request should be authorized.

5 The Semantic Foundation of RT: Constraint Datalog

The original design of the RT framework, as presented in [19], was based on DATALOG. DATALOG is a restricted form of logic programming with variables, predicates, and constants, but without function symbols. Several previous TM languages are based on DATALOG, e.g., Delegation Logic [17], SD3 [13], and Binder [10]. DATALOG is attractive because of the following reasons.

1. DATALOG is declarative and is a subset of first-order logic; therefore, the semantics of a DATALOG-based TM language is declarative, unambiguous, and widely understood.
2. DATALOG has been extensively studied both in logic programming, and in the context of relational databases as a query language that supports recursion. TM languages based on DATALOG can benefit from past results and future advancements in those fields.
3. The function-symbol-free property of DATALOG ensures its tractability. For a safe DATALOG program with a fixed upper bound on the number of variables per rule, construction of its minimal model takes time polynomial in the size of the program.
4. There are efficient goal-directed evaluation procedures for answering queries.

However, DATALOG has limitations as a foundation of TM languages. One significant limitation is the inability to describe structured resources. For example, one may want to grant permission to read the entire document tree under a given URI, assign responsibility for associating public keys with all DNS names in a given domain, restrict network connections to port numbers in a limited range, or approve routine transactions with value below an upper limit. The permission to access all files and subdirectories under a directory "/pub/rt" represents permissions to access a (potentially infinite) set of resources that seems most naturally expressed using a logic programming language with function symbols. However, the tractability of DATALOG is a direct consequence of the absence of function symbols. Previous TM languages that can express certain structured resources, e.g., SPKI, have not had a formal foundation; some studies suggest that SPKI may be ambiguously specified and intractable [12, 3].

In [18], we showed that DATALOG extended with constraints (denoted by DATALOG$^{\mathcal{C}}$) can define access permissions over structured resources without compromising the properties of DATALOG that make it attractive for trust management, thus establishing a suitable logical foundation for a wider class of TM languages. DATALOG$^{\mathcal{C}}$ allows first-order formulas in one or more constraint domains, which may describe file hierarchies, time intervals, and so on, to be used in the body of a rule, thus representing access permissions over structured resources in a declarative language.

In the rest of this section, we give a brief overview of DATALOG$^{\mathcal{C}}$. See [18] for details.

Constraint DATALOG is a restricted form of Constraint Logic Programming (CLP), and is also a class of query languages for Constraint Databases (CDB) [14, 15]. The notion of constraint database was introduced in [14], and grew out of the research on DATALOG and CLP. It generalizes the relational model of data by allowing infinite relations that are finitely representable using constraints.

Intuitively, a constraint domain is a domain of objects, such as numbers, points in a plane, or files in a file hierarchy, together with a language for speaking about these objects. The language is typically defined by a set of first-order constants, function symbols, and relation symbols.

Definition 1 A *constraint domain* Φ is a 3-tuple $(\Sigma, \mathcal{D}, \mathcal{L})$. Here Σ is a signature; it consists of a set of constants and a collection of predicate and function symbols, each with an associated "arity", indicating the number of arguments to the symbol. $\mathcal{D}$ is a Σ-structure; it consists of the following: a set D called the universe of the structure, a mapping from each constant to an element in D, a mapping from each predicate symbol in Σ of degree k to a k-ary relation over D, and a mapping from each function symbol in Σ of degree k to a function from D^k into D. $\mathcal{L}$ is a class of quantifier-free first-order formulas over Σ, called the *primitive constraints* of this domain.

Following common conventions, we assume that the binary predicate symbol "$=$" is contained in Σ and is interpreted as identity in $\mathcal{D}$. We also assume that $\top$ (true) and $\bot$ (false) are in $\mathcal{L}$, and that $\mathcal{L}$ is closed under variable renaming.

The following are some examples of commonly-used constraint domains.

Equality constraint domains The signature Σ consists of a set of constants and one predicate $=$. A primitive constraint has the form $x = y$ or $x = c$, where x and y are variables, and c is a constant. DATALOG can be viewed as one specific instance of DATALOG$^{\mathcal{C}}$ with an equality constraint domain.

Order constraint domains The signature Σ has two predicates: $=$ and $<$. The Σ-structure is linearly ordered. A primitive constraint has the form $x\theta y$, $x\theta c$, or $c\theta x$ where θ is one of $=, <$. The structures in order constraint domains can be integers, rational numbers, real numbers, or some subset of them.

Linear constraint domains The signature Σ has function symbols $+$ and $*$ and predicates $\{=, \neq, <, >, \geq, \leq\}$.

A primitive constraint has the form $c_1x_1 + \cdots + c_kx_k\theta b$, where c_i is a constant and x_i is a variable for each $1 \le i \le k$, θ is any predicate in Σ, and b is a constant. Linear constraints may be interpreted over integers, rational numbers, or real numbers.

Range domains Range domains are syntactically sugared order domains. A primitive constraint has the form $x = y$, $x = c$ or $x \in (c_1, c_2)$, in which c is a constant, each of c_1 and c_2 is either a constant or a special symbol "$*$", meaning unbounded. And when c_1 is not $*$, "(" can also be "["; similarly, ")" can be "]" when c_2 is not $*$.

Tree domains Each constant of a tree domain takes the form $\langle a_1, \ldots, a_k\rangle$. Imagine a tree in which every node is labelled with a string value. The constant $\langle a_1, \ldots, a_k\rangle$ represents the node for which $a_1, \ldots, a_k$ are the strings on the path from root to this node. A primitive constraint is of the form $x = y$ or $x\theta\langle a_1, \ldots, a_k\rangle$, in which $\theta \in \{=, <, \le, \prec, \preceq\}$, $x < \langle a_1, \ldots, a_k\rangle$ means that x is a child of the node $\langle a_1, \ldots, a_k\rangle$, and $x \prec \langle a_1, \ldots, a_k\rangle$ means that x is a descendant of $\langle a_1, \ldots, a_k\rangle$.

Tree domains can be used to represent hierarchical resources such as file systems, DNS names, etc. Example 2 uses tree domains and range domains to represent network permissions. There, instead of using $\prec$, we use the value set constructor "descendants" as a syntactic sugar.

Definition 2 Let Φ be a constraint domain.

- A *constraint k-tuple*, or a *constraint*, (in variables $x_1, \ldots, x_k$) is a finite conjunction $\phi_1 \wedge \cdots \wedge \phi_N$, where each $\phi_i, 1 \le i \le N$, is a primitive constraint in Φ. Furthermore, the variables in each ϕ_i are all free and among $x_1, \ldots, x_k$.
- A *constraint* (DATALOG) *rule* has the form:

$$\begin{aligned} R_0(x_{0,1}, \ldots, x_{0,k_0}) \quad :- \quad & R_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, \\ & R_n(x_{n,1}, \ldots, x_{n,k_n}), \psi_0 \end{aligned}$$

  where ψ_0 is a constraint in the set of all variables in the rule. When $n = 0$, the constraint rule is called a *constraint fact*.

Under certain conditions, a constraint rule with n hypotheses can be applied to n constraint facts to produce some new constraint facts, somewhat similar to applying a DATALOG rule to DATALOG facts. The process of applying a rule to a set of facts requires a form of quantifier elimination. The least fixpoint of a DATALOG$^{\mathcal{C}}$ program over any constraint domain that admits quantifier elimination may be computed by iterated rule application. We say a constraint domain is tractable if it roughly has the same complexity as the equality constraint domain, which is the one used in DATALOG.

In TM languages, it is useful to appeal to constraints from several domains. It is straightforward to define multi-sorted DATALOG$^{\mathcal{C}}$, following the standard definition of multi-sorted first-order logic. In order to keep each constraint domain separate from the others, we assume that when constraint domains are combined, each domain is given a separate sort, all predicate symbols are only applicable to arguments from the appropriate constraint domain, and each variable belongs to only one sort. A multi-sorted DATALOG$^{\mathcal{C}}$ program with constraints in several domains can be evaluated in time polynomial in the size of the program if all involved constraint domains are tractable.

The translation from RT credentials to DATALOG, as presented in [19], can be used to translate RT credentials that use constraint-based value sets to DATALOG$^{\mathcal{C}}$.

6 Distributed Credential Chain Discovery

In [20], goal-directed credential chain discovery algorithms for RT_0 was presented. These algorithms also work when credential storage is distributed. In [25], a trust negotiation protocol that use RT_0 and the chain discovery mechanism in [20] was presented. This protocol enables two parties to do joint chain discovery in an interactive manner; it also enables the parties to use policies to protect sensitive credentials and the attribute information contained in the credentials.

Distributed storage of credentials raises some nontrivial questions for chain discovery. When trying to construct a credential chain to answer an access-control query, where should one look for credentials? Often, one cannot look everywhere; in that case, when a chain cannot be found, how can one be sure that none exists? Distributed credential chain discovery requires a scheme to address these questions.

Example 6 Consider the following credentials from Example 1.

$$\begin{array}{llr} \text{EOrg.preferred} \longleftarrow \text{EOrg.university.student} & & (1) \\ \text{EOrg.university} \longleftarrow \text{ABU.accredited} & & (2) \\ \text{ABU.accredited} \longleftarrow \text{StateU} & & (3) \\ \text{StateU.student} \longleftarrow \text{Alice} & & (4) \end{array}$$

These four credentials prove that Alice is a member of EOrg.preferred. When these credentials are stored in a distributed manner, it is non-trivial to guarantee that this chain can always be discovered. For example, when both (2) and (3) are stored by ABU, then just knowing (1) and (4), one does not know where to look for credentials. Also, one wants to avoid having to go to every university one by one to determine whether Alice is a university student.

In [20], a storage type system and a notion of well-typed credentials were presented to address these problems. They guarantee that credential chains can be discovered even when credentials are stored in a distributed manner. The types also guide search in the right direction, avoiding huge fan-outs.

We now give an summary of the storage type system. Each credential is assumed to be stored by its issuer (an *issuer-stored* credential) or by its subjects (a *subject-stored* credential), where subject and issuer are defined as follows. For a credential, $A.r \longleftarrow e$, we call A the *issuer*, and each principal in *base*(e) a *subject* of this credential, where *base*(e) is defined as follows: *base*$(D) = \{D\}$, *base*$(B.r_2) = \{B\}$, *base*$(A.r_1.r_2) = \{A\}$, *base*$(A_1.r_1 \cap \cdots \cap A_k.r_k) = \{A_1, A_2, \ldots, A_k\}$. Each role name has two types: an issuer-side type and a subject-side type. On the issuer side, the possible types are issuer-(traces-)none, issuer-(traces-)def, and issuer-(traces-)all, in order of increasing strength. If r is issuer-def or issuer-all, each credential of the form $A.r \longleftarrow e$ is required to be issuer-stored. If r is issuer-all, the well-typing rule for credentials additionally requires e to be issuer-all. (The types for role expressions are determined from role names.) This means that from A, one can retrieve the credential and discover its subject, allowing one to find all issuer stored credentials issued by that subject, and to repeat the process to discover all members of $A.r$. On the subject side, the possible types are subject-(traces-)none and subject-(traces-)all. If r is subject-all, each credential of the form $A.r \longleftarrow e'$ is required to be subject-stored, and e' must also be subject-all. To ensure that credentials are either issuer-stored or subject-stored, the well-typing rule also requires that no role expression is both issuer-none and subject-none. A linked role $A.r_1.r_2$ is issuer-all when both r_1 and r_2 are; same is true for subject-all. $A.r_1.r_2$ is issuer-def if r_1 is issuer-def and r_2 is subject-all, or r_1 is issuer-all and r_2 is issuer-def.

For example, a typing for credentials in Example 6 is to make preferred and university issuer-def and subject-none, and to make accredited and student issuer-none and subject-all. To make these credentials well-typed, credentials (1) and (2) should be stored by their issuer, EOrg, (3) should be stored by its subject StateU, and (4) stored by Alice. This arrangement enables our search algorithm to find all the credentials starting from EOrg and Alice, without touching credentials about other students and other universities.

7 Implementation and Applications

A Java-based RT_0 inference engine is implemented. This engine is used in two demonstration applications to be described below. It is also used in the implementation of the Trust Target Graph trust negotiation protocol developed under the Attribute-Based Access Control project. We are in the process of implementing an inferencing engine for RT_1, which will provide a more expressive basis for these and other applications.

7.1 August: Secure Distributed Calendar

August is a distributed calendar program. In August, each user has a calendar and can specify policy that determines who is allowed to view each part of the user's calendar and who is allowed to add an activity of certain kinds at a certain time. In effect, August is a simple distributed database, where the data is in a special format used for scheduling information.

Each August user declares different roles (or groups), e.g., friends, family members, colleagues, etc., and defines the members of these roles. Users may use delegation in defining role members. For example a user may specify that "friends $\longleftarrow$ my family members' friends". This avoids the need to explicitly list all the eligible users and decentralizes the assignment of roles to appropriate authorities.

Each calendar consists of activities. Each activity has a category, a time period, an importance level, and a creator, in addition to other information about the specific activity. A user can define calendar categories and the time periods in which activity of each category can be scheduled. A user can assign read and write permissions to roles. A read permission is parameterized by a category. A write permission is parameterized by a category and an importance level, which limits the maximum important level someone who possesses this permission can schedule. An August user sets policy using dialog boxes, with policy preferences translated into RT. These RT policy statements associated with August thus determines who can read or write to another user's calendar entry.

7.2 U-STOR-IT: Secure Web-based File Sharing

The Digital U-STOR-IT distributed file-sharing application is a web-based file sharing system formulated to provide a useful service and to allow us to experiment with policy development and policy requirements.

A U-STOR-IT user connects to the service using a browser, with user authentication done using client-side certificates generated by the U-STOR-IT certificate authority. Access is controlled by policy; access to an individual locker or file within a locker is determined by policy statements specified by several users. The policy language also allows one user to refer to the policy of another user when specifying his policy.

In addition to storing files, U-STOR-IT has message processing and version control facilities, providing a secure collaboration tool centered around policy concepts expressed in RT.

8 Service Selection and Mobile Code

The Stanford architecture for mobile code security is based on enhancements to the Jini primitives and conventions for calling library services. When clients want to access services, they lookup proxies in directories and then use proxies in ways similar to a local service. This is more flexible than traditional client/service architecture in that it is not just data that are transferred between clients and servers. However, this approach involves mobile code risks and the need to manage trust in code obtained from services not previously know to the client. RT provides a natural way of expressing trust policies in this environment.

The four security goals of the mobile code architecture are: (1) Restrict the ability of proxy code to access to the client virtual machine on which it executes, (2) Secure the proxy-service network communication, (3) Allow the client to authenticate the proxy and make sure that it comes from the right service, (4) Allow the service to authenticate and authorize a requesting client. Goal (1) is addressed by Java security mechanisms. In addition, a bytecode filtering mechanism can be used to enhance the protection. A Jini client manages mobile code risks by invoking a Java bytecode filter that screens Java bytecode for security risks. The filter may be customized to enforce coding restrictions or software conventions established by a coalition and designed to prevent mobile code attacks. Further information about the Java bytecode filtering mechanisms used by a client to examine a service proxy before it is installed in the virtual machine may be found in [8]. Goal (2) may be achieved by consistent use of a communication protocol such as SSL (Secure Sockets Layer; standard implementations of Java RMI over SSL may be used). Our focus is therefore to provide additional security mechanisms to meet goals (3) and (4). For goal (3), we need to authenticate code and data. For goal (4), we wish to enable single authentication per session, multiple authentication mechanisms, and authorization based on the arguments of calls.

The main mechanisms for meeting this goals are a new form of signed service proxy that includes both signed code and signed object state, and a form of authenticated service session. Signed service proxies with state (such as the IP address used to obtain the service) allow client programs to verify a signature on a proxy before making any call on the proxy. When a client gets a proxy from the directory, it first verifies that the proxy is signed by a public key, and then consults a trust-management engine to ensure that the signing public key is trusted. In an authenticated client session, when a client program makes the first call to the service, the service consults a TM engine (on the service side) to compute a set of permissions to grant the client. The service generates a dedicated service session, associates the set of permissions with this session, generates a proxy for the dedicated service session, and returns the proxy to the client. Since the service session and the session proxy are dynamically generated and transmitted through a secure channel, they can use secret-key based secure communication channel. The dedicated service session provides performance optimization and controls clients' use of services in a flexible, policy-driven manner.

9 Conclusions

We have presented an overview of RT, a Role-based Trust-management framework. Our presentation summarizes main aspects of RT described in previous publications, together with recent improvements and explanation of motivating applications. The RT framework provides policy language, semantics, deduction engine, and pragmatic features such as application domain specification documents that help distributed users maintain consistent use of policy terms. In comparison with systems like SPKI/SDSI and KeyNote, the advantages of RT include: a declarative, logic-based semantic foundation, support for distributed chain discovery and vocabulary agreement, strongly-typed credentials and policies, more flexible delegation structures, and more expressive support for Separation-of-Duty policies.

References

[1] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, October 1993.

[2] Andrew W. Appel and Edward W. Felten. Proof-carrying authentication. In *Proceedings of 6th ACM Conference on Computer and Communications Security (CCS-6)*, November 1999.

[3] Olav Bandmann and Mads Dam. A note on SPKI's authorization syntax. In *Pre-Proceedings of 1st Annual PKI Research Workshop*, April 2002. Available from http://www.cs.dartmouth.edu/˜pki02/.

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

[5] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote trust-management system, version 2. IETF RFC 2704, September 1999.

[6] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, May 1996.

[7] Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in XML. W3C Recommendation, January 1999.

[8] Ajay Chander, John C. Mitchell, and Insik Shin. Mobile code security by Java bytecode instrumentation. In *DARPA Information Survivability Conference & Exposition (DISCEX II)*, June 2001.

[9] David D. Clark and David R. Wilson. A comparision of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 184–194. IEEE Computer Society Press, May 1987.

[10] John DeTreville. Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 105–113. IEEE Computer Society Press, May 2002.

[11] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI certificate theory. IETF RFC 2693, September 1999.

[12] Jonathan R. Howell. *Naming and sharing resources acroos administrative boundaries*. PhD thesis, Dartmouth College, May 2000.

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

[14] Paris C. Kanellakis, Gabriel M. Kuper, and Peter Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, August 1995. Preliminary version appeared in *Proceedings of the 9th ACM Symposium on Principles of Database Systems (PODS)*, 1990.

[15] Gabriel Kuper, Leonid Libkin, and Jan Paredaens, editors. *Constraint Databases*. Springer, 2000.

[16] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.

[17] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation Logic: A logic-based approach to distributed authorization. *ACM Transaction on Information and System Security (TISSEC)*, February 2003. To appear.

[18] Ninghui Li and John C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages*, January 2003. To appear.

[19] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[20] Ninghui Li, William H. Winsborough, and John C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, February 2003.

[21] Ronald L. Rivest and Bulter Lampson. SDSI — a simple distributed security infrastructure, October 1996. http://theory.lcs.mit.edu/˜rivest/sdsi11.html.

[22] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[23] Tichard T. Simon and Mary Ellen Zurko. Separation of duty in role-based environments. In *Proceedings of The 10th Computer Security Foundations Workshop*, pages 183–194. IEEE Computer Society Press, June 1997.

[24] William H. Winsborough and Ninghui Li. Protecting sensitive attributes in automated trust negotiation. In *Proceedings of ACM Workshop on Privacy in the Electronic Society*, November 2002. To appear.

[25] William H. Winsborough and Ninghui Li. Towards practical automated trust negotiation. In *Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks (Policy 2002)*, pages 92–103. IEEE Computer Society Press, June 2002.