Role-based Privilege and Trust Management

Dongwan Shin[1] and Gail-Joon Ahn[2]

[1] Department of Computer Science
New Mexico Tech, 801 Leroy Place
Socorro, NM 87801, USA
doshin@nmt.edu

[2] Department of Software and Information Systems
University of North Carolina at Charlotte
Charlotte, NC 28223, USA
gahn@uncc.edu

Abstract. The Internet provides tremendous connectivity and information sharing capability which organizations can use for their competitive advantage. However, we still observe security challenges in Internet-based applications, especially in terms of their limited support for controlled access to organizational resources and information for *unknown* users. Roles can be a convenient construct for expressing entitled privileges and trust degree alike, based upon which further specification of responsibility and capability is made so as to facilitate trust-based authorization for such an environment. In this article, we design a role-based privilege and trust management by leveraging a role-based trust model and a privilege management infrastructure, as an attempt to develop an easy-to-use, flexible, and interoperable authorization mechanism for unknown users. Also, we demonstrate the feasibility of our mechanism by providing a proof-of-concept prototype implementation using commercial off-the-shelf technologies.

1 Introduction

The Internet is uniquely and strategically positioned to address the needs of a growing segment of population in a very cost-effective way. It provides tremendous connectivity and immense information sharing capability which organizations can use for their competitive advantage. However, we still observe security challenges in this open environment, especially in terms of controlled access to organizational resources and information for unknown users.

Conventional access control policies rest on the principle that a user requesting access should be known a priori. As a result, it is assumed that the user's identity should exist within policy-effective domain(s) where an access control decision can be made. Discretionary access control (DAC), mandatory access control (MAC), and role-based access control (RBAC) all remain in line with the principle; they regulate access based on the known identity or some attributes associated with it such as groups, security clearances, and roles. We call this type of attribute assignment *identity-based attribute assignment (IAA).*

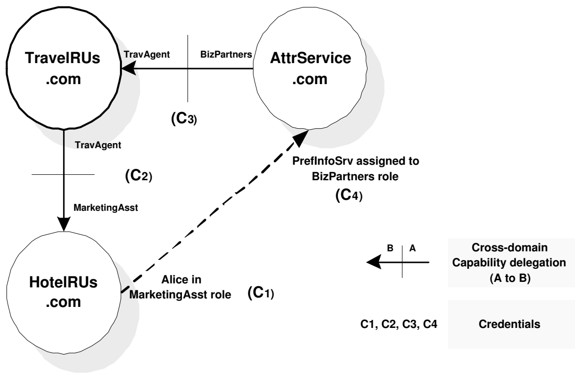

Fig. 1. An example illustrating the usage of roles for capability delegation in cross-domain environments

However, those conventional access control approaches based on IAA alone are often inadequate to meet all the requirements that today's Internet environments usually set out [15, 24]. This is especially true when the aforementioned principle does not hold any more.

Trust management (TM) has attracted growing attention as a generalized access control mechanism for lowering the bar on the limitation of conventional access control approaches. Central to understand the concept of TM is capability delegation, which is incorporated into a credential chain from a resource owner to an access requester; access is granted if a chain of credentials proves that a requested action complies with local access control policies. Hence, it provides an authorization framework for unknown users, whereby it is possible to express and evaluate decentralized access control policies and credentials in an open and distributed network environment such as the Internet [5].

1.1 A Motivating Example

Considering their indirect and bilateral characteristics, roles can be a convenient construct enabling trust-based authorizations. Roles provide an indirection mechanism for relating users with permissions, thereby reducing complexity and potential errors in permission management. In addition, roles can represent abilities and groups of trusted users in cross-domain environments, which often derive from bilateral agreements between organizations. Let us walk through a simple example aimed at illustrating the usage of roles in a cross-domain environment, and we will use it throughout this paper.

Example 1. Assume that a company AttrService.com provides its customers' travelling preferences information to its business partners through a PrefInfoSrv service; a permission to access PrefInfoSrv is associated with its BizPartners role (permission assignment or PA in RBAC jargon)[3]. As a result of its recent part-

[3] We assume that access control policies in the example are based on RBAC.

nership agreement with a company TravelsRUs.com, AttrService.com allows the members of TravAgent role in TravelsRUs.com to have the permission to access PrefInfoSrv. Additionally, AttrService.com consents the permission to be further delegated by TravelsRUs.com. As a result, TravelsRUs.com further delegates the delegated permission of its TravAgent role to the MarketingAsst role in its subsidiary, HotelsRUs.com.

Assuming that *Alice* is an employee of the company HotelsRUs.com and her job title is MarketingAsst, how can she be allowed to access PrefInfoSrv based on conventional RBAC approaches? Since AttrService.com does not have *Alice* in its user-role assignment database, access control approaches relying on IAA alone cannot be much helpful in this situation. Instead, a possible scenario in TM approach could be like this; upon *Alice*'s access request, AttrService.com may request her of a chain of credentials which can prove that she is authorized to access the service. For instance, the chain may include C_1 that states *Alice* is a member of MarketingAsst in HotelsRUs.com (user assignment or UA in RBAC jargon), C_2 that the capability of TravAgent in TravelsRUs.com is delegated to MarketingAsst, C_3 that the capability of BizPartners in AttrService.com is delegated to TravAgent, and C_4 that the permission to access PrefInfoSrv is associated with the role of BizPartners within AttrService.com (permission assignment or PA in RBAC jargon). Therefore, the chain (C_1, C_2, C_3, C_4) helps AttrService.com make an access control decision for Alice's request. This is illustrated in Figure 1.

1.2 Our Objective

There has been extensive research on modelling roles for access control in various dimensions. Interestingly, however, previous RBAC models lend themselves to the primary use in closed and centralized network environments [10, 22], and thus do not address a trust-relevant aspect of roles, i.e., suitable for use as the representation of trust degree. For instance, either the trust relation or the cross-domain authority delegation shown in C_2 and C_3 is hard to express in RBAC jargon[4]. Hence, it would be desirable to find a component that correlates with the notion of trust on RBAC. Given such a component, RBAC can be more flexible and suitable for use in open and decentralized network environments. In this light, we view our contribution as follows.

- With the introduction of such a component, we improve current RBAC models to support the notion of trust so that the models can natively support access control for unknown users.
- We introduce a reference model for designing roles or comparing different designs of roles in the authorization framework that TM provides.
- We design a mathematical framework in order to instantiate our model by accommodating an existing framework.

[4] Strictly speaking, role-based delegation is analogous, if not identical. However, most discussions on role-based delegation in the literature have been limited within the context of easing administrative burdens in a single domain.

– We design an authorization mechanism and implement a security architecture based on our model in order to address access control for unknown users.

Our primary objective in this paper is 1) to present a formal description of a trust-enabled RBAC called TRUSTr, and 2) to design a role-based privilege and trust management by leveraging both the model and the privilege management infrastructure (PMI), as an attempt to develop an easy-to-use, flexible, and interoperable role-based authorization mechanism for open and distributed environments. TRUSTr introduces a new component called trust assignment (*TA*). TA features trust-based role association among different administrative domains and supports entrusting or distrusting operations for the management of associated roles. We discuss role hierarchies and constraints in conjunction with TA for their varying semantics. PMI has been introduced in order to support a simplified privilege management among distributed Internet-based applications by leveraging X.509 attribute certificates [12, 23]. It enables us to establish the trustworthiness among different authorization domains as long as each of them keeps the meaning of attributes intact.

The rest of this paper is organized as follows. Section 2 shows background technologies and previous research related to our work. Section 3 presents our TRUSTr model, followed by Section 4 which describes our approach to designing the role-based privilege and trust management. Implementation details are described in Section 5. Section 6 discusses lessons learned from our experiment and concludes the paper.

2 Background and Related Works

In this section, we start with the discussion of TM in the context of distributed environments. Then we discuss previous approaches to using roles in TM systems or certificate-based authorization systems.

2.1 Trust and Trust Management

A number of TM systems or languages have been implemented or proposed over the last decade: PolicyMaker, KeyNote, REFEREE, SPKI/SDSI, TPL, and more recently RT [4–6, 8, 11, 14]. In addition, a mathematical framework for better understanding those TM systems was proposed in [25]. The framework describes how to model different types of authorization in a consistent manner by using a lattice structure, how to model different kinds of credentials by using a monotone license function, and how to implement TM engines by using the least fixpoint semantics. As shown in the upper part of Table 1, the framework consists of five elements; principals, authorizations, authorization maps, licenses, and assertions. A principal is an entity that makes or authorizes access requests. An authorization pertains to the permission granted by a principal. Authorizations are organized as a lattice, which is an important characteristic that enables

$p \in Principal$
$u \in Auth$
$m \in AuthMap \; = \; Principal \rightarrow Auth$
$l \in License \; = \; AuthMap \rightarrow_m Auth$
$a \in Assertion \; = \; Principal \times License$

$\mathcal{M}_{Assertions} : \mathcal{P}(Assertion) \rightarrow_m AuthMap$
$\mathcal{M}_{Assertions}(A) \; = \; lfp(\lambda m.\lambda p. \bigsqcup\{l(m) | \langle p, l\rangle \in A\})$

$\mathcal{M}_{Engine} : Principal \times Auth \times \mathcal{P}(Assertion) \rightarrow Bool$
$\mathcal{M}_{Engine}(p, u, A) \; = \; u \sqsubseteq \mathcal{M}_{Assertions}(A)(p)$

Table 1. Weeks' Framework for understanding TM Systems

multiple authorizations made by the same principal into a single authorization. An authorization map is a function which associates a principal with an authorization. A license is a monotone function (denoted by $\rightarrow_m$) which associates an authorization map with an authorization. The monotonicity means that authorizations can only increase by adding other principals' authorizations in an authorization map. An assertion pertains to an authorization expression made by a principal, which can be viewed as credentials or certificates.

The lower part of Table 1 describes the semantics of assertions and TM engines to handle authorization decisions. The semantics of assertions, denoted by $\mathcal{M}_{Assertions}$, is a function which associates a set of assertions with an authorization map. Simply put, given a set of assertions made by various principals, $\mathcal{M}_{Assertions}$ returns a coherent authorization map that contains the authorization granted by each of the principals. A least fixpoint, denoted by lfp, is taken to find such an authorization map[5]. A TM engine makes an access control decision on the basis of an authorizing principal, a request, and a set of assertions. The engine computes an authorization map from the set of assertions, and determines if the request is less than authorizations granted by the authorizing principal. As we shall discuss it later in this paper, we use this framework for the purpose of describing a formal instantiation scheme of our proposed model and approach.

2.2 Roles in Trust Management Systems

The usage of roles for access control can be found in various TM systems. One of them is ISO/IEC's PMI, based on global namespace, utilizing X.509 attribute certificate framework [7, 9, 12, 23]. PMI is an extension of public key infrastructure (PKI) in the light of authorization. The attribute certificate binds entities

[5] λ in Table 1 is used as a function in $A \rightarrow B$ in which $\lambda a.e$, given a where $a \in A$, returns the result of expression e where $e \in B$.

to attributes such as roles or groups. Along with attribute certificates, PMI is introduced with its four models: general model, control model, delegation model, and roles model. General and control models are required, whereas roles and delegation models are optional. The general model provides the basic entities which recur in other models. On the other hand, SPKI/SDSI [8, 20] is another TM system, based on local namespace, supporting the usage of roles.

Recently, the practicality of SPKI/SDSI's capability-style approach to granting permissions has been questioned by Li et al [14]. They argued that capability-style TM systems generally lack the expressive power for role-based or group-based authorization policies, and thus they frequently result in heavy administrative burdens on public key management and distribution. As an alternative, they proposed RT framework, a family of role-based TM languages which are able to express policies and credentials for distributed authorizations. Although the RT framework provides the rich expressive power for authorization policies and authority delegation, it still needs to address some important issues. One of them is that the distinction between a user and a principal (generally public key) is not clear in their framework, and thus it seems to be difficult to express some constraints that should be applied to the user.

There have been also rigorous researches on how to use attribute certificates with the view of managing privileges on distributed systems. In the OSF/DCE environment [16, 17], privilege attribute certificate (PAC) that a client can present to an application server for authorization was introduced. PAC provided by a DCE security server contains the principal and associated attribute lists, which are group memberships. The application server works as a reference monitor to make access control decisions based on the comparison between the client's attributes and attributes in ACLs. This approach focused on the traditional group-based access control. Similarly, Thompson et al. [24] developed a certificate-based authorization system called Akenti for managing widely distributed resources. It was especially designed for system environments where resources have multiple stakeholders and each stakeholder wants to impose conditions for access. There are two types of certificates employed for authorization: use-condition certificate and attribute certificate. The stakeholders assert their access requirements in use-condition certificates and an attribute authority issues attribute certificates that bind a user to attributes. Their approach emphasized the policy-based access control in a distributed environment.

Also, several studies have been carried out to make use of RBAC features with the help of public-key certificates [3, 18, 19]. Public-key certificates were used to contain attribute information such as role in their extension field. Two architectures have been identified in [19]: user-pull and server-pull. [3] demonstrated how RBAC can be injected to secure a web-based workflow system using the user-pull style architecture whereas [18] described the server-pull style architecture for role-based authorization, adopting an LDAP-oriented approach. To add role information into public key certificates, however, may cause problems such as shortening of certificates' lifetime and complexity of their management. A user's role memberships are dynamic entities even though roles themselves are persistent, compared to the user's identity. Whenever role memberships change, a new public key certificate binding the user's identity and new roles needs to be issued. Subsequently, it leads unnecessary revocation of a public-key certificate which could be still valid for identity affirmation purposes.

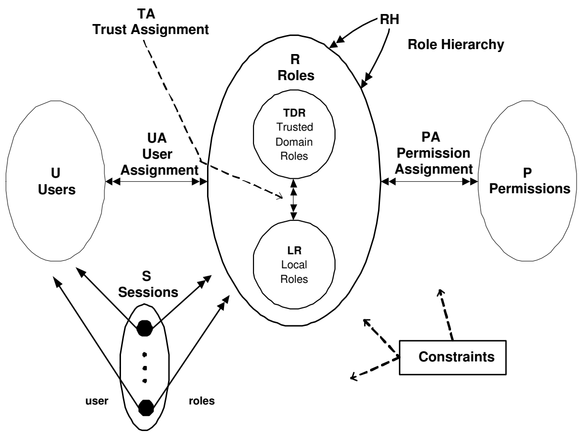

Fig. 2. The TRUSTr model as an extension of RBAC96 model

3 The Model

In this section, we discuss an extension of RBAC models, called TRUSTr, and its formal definition. With a view to leveraging roles as a convenient construct for expressing the degree of trust in trust-based authorization, we introduce a new component called trust assignment (*TA*).

3.1 Basic Components

As shown in Figure 2, TRUSTr subsumes existing RBAC components from [22], also following the conventional approach to defining them using the sets, relations, and functions.

Definition 1. *The basic components of TRUSTr consist of existing RBAC components as follows.*

- U, R, P, and S, representing the set of users, roles, permissions, and sessions, respectively.
- UA, PA, and RH, representing the relation of user-to-role assignment, permission-to-role assignment, and role hierarchy, respectively. RH is partial order on R, written as $\preceq$.
- user: $S \rightarrow U$ *represents a function mapping each session* s_i *to the single user.*

- roles: $S \rightarrow 2^R$ represents a function mapping s_i to a set of roles, where roles $\subseteq \{r|(\exists r^{'} \succeq r)[(user(s_i), r^{'}) \in UA]\}$ and s_i has permissions $\bigcup_{r \in roles(s_i)}\{p|(\exists r^{''} \preceq r)[(p, r^{''}) \in PA]\}$.

3.2 Trust Assignment

Similar to assignment relations discussed above, trust assignment (*TA*) can be described as a trust-to-role assignment relation. However, the relation does not seem to be self-explanatory, since the trust *assignable* to roles is undefined yet. Hence, we 1) define what the assignable trust is, 2) identify components appropriate for representing the trust, and 3) re-define *TA* based 1) and 2).

We first define authority domains. Each of authority domains in our model represents a single RBAC domain where RBAC administration and enforcement can take place.

Definition 2. *Let D denote a set of authority domains. $D = \{d_1, ..., d_o\}$.*

In our model, the degree of trust determines the amount of capability delegable from one authority domain to another. That is, how much d_i can trust d_j can be defined as how much capability to do some tasks within d_i is given to d_j by d_i. Since roles generally represent capability, we believe that roles and role hierarchy are a good means to specify the degree of trust; since roles are hierarchically organized (mathematically partially ordered), their hierarchy can be naturally viewed as such. We call these roles local roles (*LR*), and *LR* is used to represent the authority to be delegated to other authority domains. On the other hand, roles in trusted authority domains are useful in two aspects; they help avoid listing all users in the trusted authority domains that will be given the delegated authority and also facilitate cascading authority delegation. We call these trusted domain roles (*TDR*). Central to our model is the idea of trust-based role association between *LR* and *TDR*, which enables authority delegation from a local domain to trusted domains. This is backed by TA, which is shown in Figure 2.

Definition 3. *Roles R are either local roles (LR) or trusted domain roles (TDR). Suppose TDR_{d_i} denotes the subset of trusted roles from authority domain d_i in the local domain, $TDR = \bigcup TDR_{d_i}$, where $i = 1, ..., n$. We let $R = LR \cup TDR$ denote the set of all roles.*

Definition 4. *Trust assignment TA is a many-to-many LR to TDR assignment relation.*

We call *entrusting* a role to refer to the association of an element of *LR* with an element of *TDR*. Similarly, we call *distrusting* a role to refer to the break-up of their association. Roles can entrust or can be entrusted by other roles only across authority domains. That is to say, a role cannot entrust another role in the same authority domain. We consider trust assignment relation to be role dominance relation but in a reverse order. For the role entrusting and distrusting operations, we use the notation $A \Rightarrow_C B$ to represent that role A entrusts role B under the condition of satisfying a set of constraints C, and $A \not\Rightarrow B$ to represent that role A distrusts role B. $A \Rightarrow_C B$ means that role B is at least as powerful as role A; role B inherits all or partial permissions associated with role A depending upon C, and role A inherits all users assigned to role B. It should be noted that one type of constraints in C may specify the degree of authority delegation on role B. We will discuss different types of constraints in a later section.

Definition 5. *Suppose we are given the relation $\mathsf{A} \Rightarrow_C \mathsf{B}$, we call A as an entrusting role and B as an entrusted role. Similarly, when we are given $\mathsf{A} \not\Rightarrow \mathsf{B}$, we call A as a distrusting role and B as a distrusted role. Local roles cannot be either an entrusted role or a distrusted role.*

TA involves two types of assignment relation: *explicit* and *implicit*. Trust assignment is explicit when a role from LR entrusts a role from TDR. On the other hand, trust assignment can be made implicitly between LR and TDR_{d_j} when a role from TDR_{d_i}, which is entrusting a role from TDR_{d_j}, is entrusted by a role from LR, where domains d_i and d_j are different.

Definition 6. *As we discussed earlier, trust relation is a role dominance relation, i.e., a partially ordered set. Suppose there is a trust assignment, $\mathsf{ta} = (\mathsf{lr}, \mathsf{nlr})$ where $\mathsf{lr} \in \mathsf{LR}$, $\mathsf{nlr} \in \mathsf{TDR}$, and $\mathsf{lr} \Rightarrow_C \mathsf{nlr}$. We say ta is explicit if and only if lr is covered by nlr, i.e., $\mathsf{lr} \Rightarrow_C \mathsf{x}$ and $\mathsf{x} \Rightarrow_C \mathsf{nlr}$ implies $\mathsf{lr} = \mathsf{x}$. Otherwise, ta is implicit.*

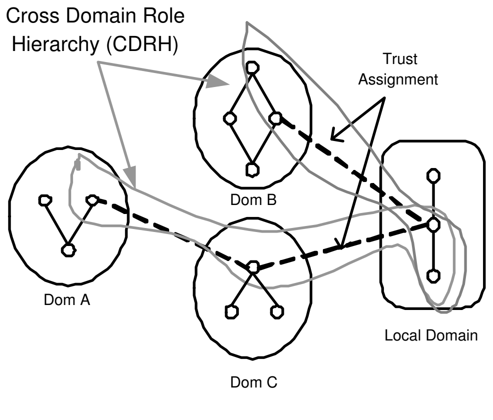

Fig. 3. Cross domain role hierarchy (CDRH) constructed by trust assignment (TA) and examples of its path

3.3 Role Hierarchies with Trust Assignment

Since *TA* introduces another type of role dominance relation, the need for its discussion in terms of *RH* is apparent when we come to deal with its semantics. We call this as a *trust* aspect of role hierarchy. We also refer to this as cross domain role hierarchy (*CDRH*)[6]. Figure 3 shows *CDRH* which is constructed as a result of *TA*. We believe that the inclusion of *CDRH* into *RH* as distinct is unnecessary since *CDRH* is the subset of the union of *TA* and *RH*. However, *CDRH* deserves further discussion for its merits; it is *dynamically* constructed in an order-preserving way, as the determination of its path is dependent upon the discovery of credentials expressing TA. In Figure 3, there are two distinct cross domain role hierarchies. One encompasses an entrusted role, its senior role in domain B and an entrusting role, its junior role in local domain, while the other covers an entrusted role (*by an entrusting role in domain C*) in domain A, an entrusted role (*by an entrusting role in local domain*) in domain C, and the entrusting role, its junior role in local domain. For instance, assume that Alice in our earlier example is the member of the entrusted role in domain A, she can take up two roles in the local domain in *CDRH* by presenting a chain of credentials similar to the one that we discussed in the earlier section.

In TRUSTr, role hierarchies are extended to trusted domains without a loss of their generality. Sandhu discusses two types of role hierarchies, permission inheritance *RH* and activation *RH* [21]. Permission inheritance *RH* concerns that a member of a senior role in the hierarchy inherits permissions from juniors, whereas activation *RH* means that a member of a senior role in the hierarchy is authorized to activate juniors for the purpose of least privilege. We already discussed *CDRH* enabling permission inheritance *RH*. *CDRH* can also be used for the purpose of the activation *RH* gracefully. However, the activation of junior roles needs to be considered only if they belong to *LR*. Role activation is closely related to a session, which associates a user with possibly many roles. It seems to be unreasonable that in such a session, a user from a trusted domain can activate roles from other trusted domains than local domain. Accordingly, *roles*, which is the component concerning the roles activated in a session, should be modified, while *user* remains unchanged.

Definition 7. *roles:* $S \rightarrow 2^R$ *is a function mapping each session* s_i *to a set of roles, where roles* $\subseteq \{r|(\exists r' \succeq r)[((user(s_i), r') \in \mathsf{UA}) \wedge (r \in \mathsf{LR})]\}$ *and session* s_i *has the permissions* $\bigcup_{r \in roles(s_i)} \{p|(\exists r'' \preceq r)[(p, r'') \in \mathsf{PA}]\}$.

3.4 Constraints with Trust Assignment

Continuing efforts have been made to identify and specify different types of constraints in RBAC models [2, 13]. In general, constraints previously identified in [2] can be used in our model without losing their original motivation and

[6] Note that the relation $\Rightarrow$ in *CDRH* could be rewritten as $\preceq$ for the sake of consistency in dominance relation, where all associated users and permissions are inherited. In such a case, $A \Rightarrow B$ is rewritten as $A \preceq B$.

intention. What we are particularly interested in this section is some types of constraints that can regulate the behavior of *TA*, and thus they can be expressed as various security policy constructs. As we discussed earlier, *TA* involves authority delegation in cross-domain environments. Therefore, we strongly believe that stricter constraints should be applied to the authority delegation enabled by *TA*. This is why we reason about some important properties related to *TA* as follows.

Bilaterality - Roles from different domains are associated based on trust relations between those domains. Our assumption is that the trust relation is established by a mutual agreement between those domains, and such an agreement should be negotiated bilaterally. two companies from the earlier example, AttrService.com and TravelsRUs.com, will agree bilaterally that AttrService delegates its BizPartners role's authority to TravelsRUs's TravAgent role.

Variability - Trust is subject to change with environmental conditions such as time and location. That is, any change in a trust relation should be reflected on the delegated authority of a role involved in the relation. For instance, assume the mutual agreement between AttrService.com and TravelsRUs.com should hold only for one year. As a result, the delegated authority of TravAgent will be valid only within the specified time frame.

Granularity - Trust varies in degree. That is, the granularity of authority delegation should be determined depending on the trust degree of a trusting domain upon a trusted one. For example, if the trust degree of AttrService.com on TravelsRUs.com's TravAgent role is high enough to allow all authority of BizPartners to be delegated to TravAgent, every permission associated with BizPartners will be delegated to TravAgent.

Transitivity - Trust relations can be transitive. That is, authority delegation using roles may be cascaded through one or more trust relations among domains. The cascading authority delegation should be managed by a controllable mechanism such as a boolean or an integer approach; the boolean control simply specifies an ability to delegate, while the integer control specifies the number of possible delegation. For instance, the trust relation between AttrService.com and TravelsRUs.com should be transitive so that the authority delegated to the TravAgent role in TravelsRUs.com can be further delegated to the MarketingAss role in HotelRUs.com.

Based upon the properties above, we can identify some possible constraints relevant with *TA*. The variability can be expressed by *temporal* or *spacial* constraints. The temporal constraint pertains to the specification of the validity period of *TA*, while the spacial constraint concerns the specification of the effective domain. For example, assume that *TA* should be constrained by both, the temporal constraint can be expressed as a variety of formats such as ValidNotBefore and ValidNotAfter, and the spacial constraint can be expressed various formats such as Targeting.

Similarly, *TA* needs to be constrained by the granularity. It is commonly believed that the level of trust is different depending on trusted entities. Granularity can be expressed as an *ordinal* constraint in trust assignment. The ordinal constraint is related to the specification of the magnitude of authority delegation in *TA*. The typical expression of the ordinal constraint might be, for example, (High, Medium, Low). This can be used for specifying further which permissions with assigned ordinal values can be delegated. On the other hand, the depth of trust is another factor having an effect on *TA*. Trust depth can be articulated by boolean control or integer control constraint, which we discussed above.

4 Designing Role-based Privilege and Trust Management

In this section, we discuss our approach to designing a role-based privilege and trust management for access control in open and distributed environments. Firstly, we start with the discussion of how we adopt attribute certificates in PMI to encapsulate role-based privileges. Then we give full details of how we define principals and identifiers based on which credentials and policies can be expressed. Note that we are using both attribute certificates and credentials interchangeably. Finally, we describe our instantiation scheme by extending Weeks' mathematical framework based on which credentials and policies can be evaluated.

4.1 Adopting Attribute Certificate

Our approach is based on basic entities in PMI. It consists of three foundation entities: the object, the privilege asserter, and the privilege verifier. The control model explains how access control is managed when privilege asserters request services on objects. When the privilege asserter requests services by presenting his/her privileges, the privilege verifier makes access control decisions based on the privileges presented and local security policies associated with those privileges. In PMI, two additional components are introduced for the purpose of articulating RBAC: role assignment and role specification. Role assignment is to associate privilege asserters with roles, and its binding information is contained in attribute certificate called role assignment attribute certificate. The latter is to associate roles with privileges, and it can be contained in attribute certificates called role specification attribute certificate or locally configured at a privilege verifier's system. Our approach is based upon and extended from these attribute certificates. We employ two additional attribute certificates along with one public key certificate, which we will discuss in the next section.

4.2 Principals, Identifiers, and Credentials

We follow the conventional approach to defining principals as entities making or authorizing access requests; principals are public keys. An authority domain is represented by a set of public keys; one or more public keys may be bound to an authority domain if they belong to the same authority domain. We typically use $K_{AttrService}$, K'_{d_i}, and K''_{d_i} for denoting specific public keys bound to authority domains. We interpret the binding between an authority domain and a public key as an assertion of a *speak-for* relation, [7], borrowed from [1]. For example, whenever a public key $K^{'}_{d1}$ speaks for d_1, that means if $K^{'}_{d1}$ makes a statement then d_1 makes the same statement. Multiple-key binding is required in some situations where, for example, public key revocation or transition is necessary. Similarly, one or more public keys are bound to a user if they belong to the same user. ; K_A, $K^{'}_{Alice}$, and $K^{''}_{Alice}$ are typically used for denoting specific public keys bound to users.

An identifier is an arbitrarily long sequence of alphabetic and numeric characters that is used to identify the following components: roles, users, and permissions. In this work, we borrowed the concept of local namespace from SPKI/SDSI [8, 20] except for compound names[8]. Roles, users, and permissions can be expressed by a principal representing an authority domain followed by their identifier whose typical example is their names. For instance, the BizPartners role from our earlier example is a local role in AttrService.com, which can be represented by $K_{AttrService}$ *BizPartners*, and the TravAgent role represented by $K_{TravelsRUs}$ *TravAgent* is a trusted domain role in AttrService.com. Similarly, users and permissions can be expressed by a sequence of length two consisting of a principal representing an authority domain followed by their corresponding identifier. For example, a user *Alice* and a permission to access PrefInfoSrv in the example can be expressed by $K_{HotelsRUs}$ *Alice* and $K_{AttrService}$ *PrefInfoSrv*, respectively.

Based on the expressions of roles, users, and permissions, five different types of credentials can be expressed as a form of $A \rightarrow B$, which can be read as "B is at least as powerful as A." Their types and syntactical examples are shown in Table 2. The semantics of all credential types except for UI are clear. The credential involving user identification (UI) can be understood as something similar to X.509 identity certificate or SPKI/SDSI's name certificate: a user's public key is bound to her identifier within an authorization domain. All credential types except for PA are used in our implementation scheme, which will be discussed in the next section.

4.3 The Scheme

The scheme, described in Table 3, should be read along with Table 1 in the previous section. The upper part of the table describes various components in TRUST*r*. It also defines the authorization lattice of TRUST*r*. The lower part of the table describes the semantics of various assertions used in the scheme.

An *ident* is a function mapping principals to users that describes users identified by each principal. For instance, $i(K_A) = Alice$ where $K_A \in Principal$ and

[7] The speak-for relation is not symmetric. That is, assuming two public keys $K^{'}_{d1}$ and $K^{''}_{d1}$ are bound to an authority domain d_1, both $K^{'}_{d1}$ and $K^{''}_{d1}$ may speak for d_1, but that does not mean that d_1, $K^{'}_{d1}$, and $K^{''}_{d1}$ are equal.

[8] Though compound names have some merits, we do not consider them in this paper since they are likely to require ontology-based name agreement between entities.

$User\ identification\ (UI)$:	$K_{HotelsRUs}\ \mathit{Alice} \rightarrow K_A$
$User\ assignment\ (UA)$:	$K_{HotelsRUs}\ \mathit{MarketingAsst} \rightarrow K_{HotelsRUs}\ \mathit{Alice}$
$Perm.\ assignment\ (PA)$:	$K_{AttrService}\ \mathit{PrefInfoSrv} \rightarrow K_{AttrService}\ \mathit{BizPartners}$
$Trust\ assignment\ (TA)$:	$K_{AttrService}\ \mathit{BizPartners} \rightarrow K_{TravelsRUs}\ \mathit{TravAgent}$
	$K_{TravelsRUs}\ \mathit{TravAgent} \rightarrow K_{HotelsRUs}\ \mathit{MarketingAsst}$
$Role\ hierarchy\ (RH)$:	$K_{HotelsRUs}\ \mathit{Employee} \rightarrow K_{HotelsRUs}\ \mathit{MarketingAsst}$

Table 2. Credential Types and Examples

$$s \in Users$$
$$e \in Permissions$$
$$r \in Roles = LRole + TDRole$$
$$\langle s, r\rangle \in UA = Users \times Roles$$
$$\langle p, r\rangle \in PA = Permissions \times Roles$$
$$\langle r^{'}, r^{''}\rangle \in TA = (LRole \times TDRole)$$
$$\langle r^{'}, r^{''}\rangle \in RH = (LRole \times LRole) + (TDRole \times TDRole)$$

$$i \in Ident \ = \ Principal \rightarrow Users$$
$$u \in Auth = Principal \rightarrow \mathcal{P}(Permissions)$$
$$\quad UAssertion = Principal \times UA$$
$$\quad RHAssertion = Principal \times RH$$
$$\quad TAssertion = Principal \times TA$$

$$\mathcal{M}_{UALicense} : UA \times AuthMap \rightarrow Auth$$
$$\mathcal{M}_{UALicense}(\langle s, r\rangle, m) = \lambda p^{'}.\{e \mid i(p^{'}) = s \ and \ \langle e = m(p)(p^{'}), r\rangle \in PA\}$$

$$\mathcal{M}_{UAssertion} : UAssertion \rightarrow Assertion$$
$$\mathcal{M}_{UAssertion}(p, \langle s, r\rangle) = \langle p, \lambda m.\{\mathcal{M}_{UALicense}(\langle s, r\rangle, m)\}\}$$

$$\mathcal{M}_{RHLicense} : RH \times AuthMap \rightarrow Auth$$
$$\mathcal{M}_{RHLicense}(\langle r^{'}, r^{''}\rangle, m) = \lambda p.\bigsqcup\{\exists\langle i(p), r\rangle \in UA,$$
$$\quad if(r^{'} \preceq r), \ then \ \mathcal{M}_{UALicense}(\langle s, r^{'}\rangle, m)(p), \ \mathcal{M}_{UALicense}(\langle s, r^{''}\rangle, m)(p)\}$$

$$\mathcal{M}_{RHAssertion} : RHAssertion \rightarrow Assertion$$
$$\mathcal{M}_{RHAssertion}(p, \langle r^{'}, r^{''}\rangle) = \langle p, \lambda m.\{\mathcal{M}_{RHLicense}(\langle r^{'}, r^{''}\rangle, m)\}\}$$

$$\mathcal{M}_{TALicense} : TA \times AuthMap \rightarrow Auth$$
$$\mathcal{M}_{TALicense}(\langle r^{'}, r^{''}\rangle, m) = \lambda p.\bigsqcup\{\exists\langle i(p), r\rangle \in UA,$$
$$\quad if(r^{'} \Rightarrow r), \ then \ \mathcal{M}_{UALicense}(\langle s, r^{'}\rangle, m)(p), \ \mathcal{M}_{UALicense}(\langle s, r^{''}\rangle, m)(p)\}$$

$$\mathcal{M}_{TAssertion} : TAssertion \rightarrow Assertion$$
$$\mathcal{M}_{TAssertion}(p, \langle r^{'}, r^{''}\rangle) = \langle p, \lambda m.\{\mathcal{M}_{TALicense}(\langle r^{'}, r^{''}\rangle, m)\}\}$$

Table 3. Scheme for implementing TRUSTr

$Alice \in User$ means that a principal K_A belongs to (or identifies) a user *Alice*. We define authorizations *Auth* in our scheme as a function lattice under the pointwise ordering, where an authorization maps a principal to a set of permissions. For instance, $K_A \rightarrow \{e_1, e_2\}$ where $e_1, e_2 \in Permission$ means that the principal K_A has the set of permissions $\{e_1, e_2\}$. For authmap m, if $\lambda K_A.\{e_1, e_2\}$ is in $m(p)$, then principal p authorizes K_A to have the set of permissions $\{e_1, e_2\}$.

We define three authorization assertion types for our scheme that can represent the abstraction of authorization certificates: *UAssertion*, *RHAssertion*, and *TAssertion*. *UAssertion* pertains to an authorization expression related to user assignment, whereas *RHssertion* expresses an authorization relevant to role hierarchy. *TAssertion* is an authorization expression concerning trust assignment. For the sake of simplicity, we consider that permission assignment is set as local policies within systems. The semantics of *UAssertion* can be understood in light of a given principal. A set of permissions is granted to the principal by a user-role relation and an authmap. For example, $\lambda K_A.\{e_1, e_2\} \in \mathcal{M}_{UALicense}(\langle Alice, r\rangle, m)$ where $\{e_1, e_2\} \subseteq Permission$ means that K_A belonging to *Alice* can have a set of permissions $\{e_1, e_2\}$ that are assigned to a role r. An *UAssertion* can be represented by $\langle p, \langle s, r\rangle\rangle$, which illustrates that a principal p authorizes a principal $p^{'}$ identifying a user s to have a set of permissions E that are assigned to a role r, where $\lambda p^{'}.E \in \mathcal{M}_{UALicense}(\langle s, r\rangle, m)$ and $E \subseteq Permission$. Given the meaning of *UAssertion*, the semantics of role hierarchy can be taken from it when certain conditions, i.e., role dominance relation, are met. The basic idea behind this is that, if there exist some user-role relations derived from a principal and if a derived role is senior to or the same that the senior role in a role hierarchy relation, then the principal can have a set of permission which is a least upper bound of two authorizations resulting from roles in the role hierarchy relation. For example, assuming $\langle e_3, r^{'}\rangle$,$\langle e_4, r^{''}\rangle \in PA$, $\lambda K_A.\{e_3, e_4\} \in \mathcal{M}_{RHLicense}(\langle r^{'}, r^{''}\rangle, m)$ where $r^{'} \succeq r^{''}$ represents that if $\langle Alice, r\rangle \in UA$ and $r^{'} \preceq r$, K_A can have a set of permissions $\{e_3, e_4\}$ which is a least upper bound of $\{e_3\}$ and $\{e_4\}$. An *RHssertion* can be represented by $\langle p, \langle r^{'}, r^{''}\rangle\rangle$, which means that a principal p authorizes a principal $p^{'}$ to have such a set of permissions if role dominance conditions are met. From a similar approach for the semantics of role hierarchy, we can give the semantics of trust assignment by substituting $\Rightarrow$ for $\preceq$. An *TAssertion* is represented by $\langle p, \langle r^{'}, r^{''}\rangle\rangle$, which means that a principal p authorizes a principal $p^{'}$ to have a set of permissions if trust assignment conditions are met.

Since we defined all the necessary components, let us see how TRUSTr trust management engine works. Suppose we want to know if principal p authorizes $p^{'}$ to perform the operation denoted by e according to the set of assertions A. We can express access request as $u = \lambda p^{'}.E \in Auth$, where $E = \{e\}$. Given the set of assertions A and the access request u, a TRUSTr trust management engine computes $\mathcal{M}_{Engine}(p, u, A)$, which is equivalent to $u \subseteq \mathcal{M}_{Assertion}(A)(p)$, which is equivalent to $E \subseteq \mathcal{M}_{Assertion}(A)(p)(p^{'})$, and determines if the request requires less permissions than is granted by p.

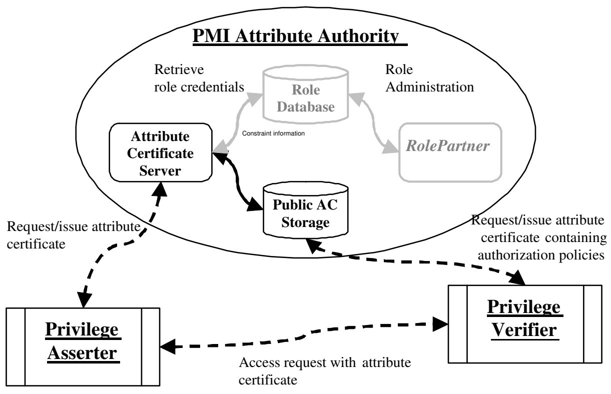
PMI Attribute Authority
Retrieve role credentials
Role Database
Role Administration
Attribute Certificate Server
Constraint information
RolePartner
Public AC Storage
Request/issue attribute certificate
Request/issue attribute certificate containing authorization policies
Privilege Asserter
Privilege Verifier
Access request with attribute certificate

Fig. 4. Operational architecture for enabling role-based privilege and trust management.

5 Implementation Details

Our implementation leverages role-based delegation enabled by TRUSTr and X.509 attribute certificate in PMI. We attempt to implement the proof-of-concept prototype implementation of our architecture. In Figure 4, three components are identified for managing attribute certificates: privilege asserter, privilege verifier, and PMI attribute authority as we discussed in the previous section. A privilege asserter is developed by using ActiveX control, named attribute certificate manager. The manager enables a user to import downloaded BER-encoded credentials into Windows registry. It also allows the user to view and select those credentials in the registry. Internet Information Server (Version 5.0) is used as a privilege verifier. An HTTP raw data filter, called AC filter, was developed using Microsoft ISAPI (Internet Server API) technology. Its main task is screening the incoming raw data from a client to see if the client presents any attribute certificate. An attribute certificate server was developed to generate attribute certificates. The programming library, called AC SDK, was built for supporting the functionality related to the generation of the attribute certificates. Netscape Directory Service 5.0 was used for both a role database and an AC storage. RolePartner was implemented in Java. It enables a role administrator in PMI attribute authority manage and configure TRUSTr components which we discussed earlier. We also developed an application working as an access control policy server. This application has been developed in C++. An engine for making access control decisions is a major component in this application.

6 Conclusion

While Internet-based applications provide organizations immense information sharing capability and connectivity, they are still confronted with security challenges, especially in light of their limited support for controlled access to organizational resources and information for unknown users. In this paper, we have discussed issues of privilege and trust management as an answer to those challenges. We also attempted to utilize a trust-enabled role model called TRUSTr and X.509 attribute certificates in PMI. In addition, we demonstrated the feasibility of our approach through a proof-of-concept implementation. We believe that this work would lead Internet-based applications to consider privilege and trust management as a core component in their design and deployment.

Acknowledgements

This work was partially supported at the Laboratory of Information of Integration, Security and Privacy at the University of North Carolina at Charlotte by the grants from National Science Foundation (NSF-IIS-0242393), and Department of Energy Early Career Principal Investigator Award (DE-FG02-03ER25565).

References

1. M. Abadi, M. Burrows, and B. Lampson. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.
2. G.-J. Ahn and R. Sandhu. Role-based authorization constraints specification. *ACM Transactions on Information and System Security*, 3(4), November 2000.
3. G.-J. Ahn, R. Sandhu, M. Kang, and J. Park. Injecting RBAC to secure a web-based workflow system. In *Proceedings of 5th ACM Workshop on Role-Based Access Control*, Berlin, Germany, July 26-27 2000. ACM.
4. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote trust-management system version 2. RFC 2704, September 1999.
5. M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 164–173, Oakland, CA, May 1996.
6. Y. H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. REFEREE: Trust management for web applications. *Computer Networks and ISDN systems*, 29(8-13), September 1997.
7. N. Dimmock, A. Belokosztolszki, D. Eyers, J. Bacon, and K. Moody. Using trust and risk in role-based access control policies. In *Proceedings of 9th ACM Symposium on Access Control Models and Technologies*, Yorktown, NY, June 2004.
8. C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. RFC 2693, September 1999.
9. S. Farrell and R. Housley. An internet attribute certificate profile for authorization. Technical report, PKIX Working Group, June 2001.
10. D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3), August 2001.
11. A. Herzberg, Y. Mass, J. Michaeli, D. Naor, and Y. Ravid. Access control meets public key infrastructure, or: assigning roles to strangers. In *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, CA, May 2000.

12. ITU. *ITU-T Recommendation X.509. Information Technology: Open Systems Interconnection - The Directory: Public-Key And Attribute Certificate Frameworks*, 2000. ISO/IEC 9594-8.
13. T. Jaeger. On the increasing importance of constraints. In *Proceedings of 4th ACM Workshop on Role-Based Access Control*, pages 33–42, Fairfax, VA, October 28-29 1999. ACM.
14. N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management. *The Journal Of Computer Security*, 11(1), February 2003.
15. J. Linn and M. Nystrom. Attribute certification: An enabling technology for delegation and role-based controls in distributed environments. In *Proceedings of 4th ACM Workshop on Role-Based Access Control*, Fairfax, VA, October 28-29 1999. ACM.
16. Open Software Foundation, Cambridge, MA. *OSF DCE 1.0 Application Development Guide*, 1992.
17. Open Software Foundation, Cambridge, MA. *OSF DCE 1.0 Introduction to DCE*, 1992.
18. J. Park, G. Ahn, and R. Sandhu. Rbac on the web using ldap. In *Proceedings of 15th Annual IFIP WG 11.3 Working Conference on Data and Application Security*, Ont., Canada, July 2001.
19. J. Park, R. Sandhu, and G.-J. Ahn. Role-based access control on the web. *ACM Transactions on Information and System Security*, 4(1), February 2001.
20. R. L. Rivest and B. Lampson. SDSI - a simple distributed security infrastructure. Technical report, September 1996.
21. R. Sandhu. Role activation hierarchies. In *Proceedings of 3rd ACM Workshop on Role-Based Access Control*, pages 33–40, Fairfax, VA, October 22-23 1998. ACM.
22. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.
23. D. Shin, G.-J. Ahn, and S. Cho. Role-based EAM using x.509 attribute certificate. In *Proceedings of Sixteenth Annual IFIP WG 11.3 Working Conference on Data and Application Security*, Cambridge, UK, July 29-31 2002.
24. M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. Certificate-based access control for widely distributed resources. In *Proceedings of 8th USENIX Security Symposium*, Washington, D.C., August 23-26 1999.
25. S. Weeks. Understanding trust management systems. In *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001.