A Solution to Distributed Delegation Chain Discovery Problem

Danfeng Yao, Roberto Tamassia

July 23, 2003

Abstract

Credential chain discovery problem in trust management is to determine whether a delegation chain exists and, if so, how to find it. In almost all the existing delegation models, individual delegations, in a form that Bob delegates to Alice, are independently made and stored, and then delegation discovery algorithms sort them out among a large credential pool and piece them into a delegation chain dynamically. The time when these individual delegations are issued may not in the same order as it in the final delegation chain. Existing delegation chain discovery algorithms dynamically look for relevant credentials from remote credential servers to construct a proof graph that may connect requester with resource owner. Because of the distributed credential storage this approach has high communication costs and slow running time, if the delegation chain is long and the credential pool is large.

We propose an alternative mechanism for delegation of authority in decentralized trust management systems. We give the notion of delegation by building *top-down* delegation chain. Top-down delegation is privilege-oriented, in which one can only delegate a right to others only if he or she is delegated it. Delegations in a single delegation chain happen sequentially. Top-down delegation makes delegation chain discovery straightforward. Delegation chain information is on a delegation credential. Each delegation is equivalent to extending the existing delegation chain by introducing a new node. All the existing nodes are the ancestor nodes of the newly added one. This implies that the information of all the entities involved in the delegation chain are on the delegation credential. The information of the entities is ordered the same order as they are delegated. So complicated delegation chain discovery algorithms [12, 33] are virtually eliminated.

Top-down delegation method makes more precise delegations than existing role-based delegation mechanism [33]. It supports privilege-based delegations and third-party delegations, and allows issuers to have tighter control over the resources. In comparison, existing role-based decentralized delegation models end up delegating privileges to ones that the delegators have no intention to.

We present *RBDD*, a decentralized role-based decentralized delegation model based on Hierarchical Identity-Based Encryption (HIBE) scheme. Applying HIBE scheme in delegation gives a natural implementation of our top-down delegation mechanism. In our model, the delegation of authority is represented as issuing a public/private key pair in HIBE scheme. A delegation public key is a human readable string that contains information of a delegation chain, which includes the identity of the original issuer and delegated entities, delegated privileges, etc. The process of delegation authentication eliminates the need of dynamic delegation chain discovery. Verifying a delegation credential is to verify whether the user has the private key corresponding to a delegation public key. The authorization process is simplified as well. The delegation chain information stored in the delegation public key is used at authorization to determine the validity of the delegation.

1 Introduction

Decentralized trust management (TM) systems are access control systems that allow initially unknown entities from different administrative domains interact and establish trust with each other through mutual

trusted entities. Several trust management systems have been proposed in recent years to address authorization issue in decentralized environments, e.g., *RT* framework [32], PolicyMaker [8], KeyNote[7, 6], SPKI/SDSI [12, 1]. The notion of delegation is essential in transferring trust and authorization in TM systems. The delegation chain connects trusted entities by the resource owner with unknown users, and plays major role at decentralized authorization. Therefore, a central problem in trust management is to determine whether such a chain exists and, if so, how to find it. This is called a credential chain discovery problem [33].

Most of existing work addressing discovery problem [8, 7, 6, 12, 1] assumes that all the potential relevant credentials are available in one place. Li, Winsborough, and Mitchell [33] are the first to identify that systems with decentralized control typically issue and often store credentials in a distributed fashion. They came up with dynamic goal-directed credential chain discovery algorithms addressing distributed storages. The algorithms dynamically collect relevant credentials from remote directories and construct a proof graph that connects requester with resource owner. This approach potentially has high communication costs and slow running time if the delegation chain is long and the credential pool is large. Forward algorithm is to construct a graph rooted by the requester to find out all the roles that the requester is a member of. It is more efficient than the backward algorithm, which constructs a proof graph rooted by a role and is to find out all members of the role.

To see how their discovery algorithms incur heavy communication overhead, consider a simple example. HospitalA gives delegations of role *guest* to the role *doctor* in a hundred other hospitals, each of whom further delegates its role *doctor* to roles of its choice in ten other institutions. Those institutions may further delegate the role and so on. When Alice wants to access resources of HospitalA as a guest and is unknown to HospitalA, a backward algorithm starts at HospitalA to find out all members of role guest. HospitalA first has to find if Alice is member of the hundred directly delegated roles. If she is, then she is granted access as a guest. Otherwise, HospitalA starts to query hundred remote servers of the hospitals, which have been delegated the guest role, to find our whom they delegate the privilege to. Each of the hundreds query results returns delegations. HospitalA asks if Alice is a member of any of the delegated roles. If there is no match for the delegated role with Alice's role, the search has to go on and HospitalA has to send queries to a thousand remote servers for information of their possible delegations.

Forward algorithm does not help with the communication overhead. Consider the following scenario. Suppose Alice has a member of five organizations, for example, she is a professor at a university, a member of ACM, etc. Each of the organizations have received a number of delegations on Alice's role made by other institutions, who may also be delegated by others and so on. In order to get access to HospitalA as a guest, she initiates a search aiming to find a delegation chain that connects her with role guest in HospitalA. She queries the directories of the five organizations to find out any delegations made to her roles by other organizations, which we refer as D. If the query results indicate that HospitalA has issued delegation of guest role, Alice then stops the search. Otherwise, she needs to query the remote servers of organizations in D until HospitalA is reached.

Clearly from the above example, the process of dynamically discovering the delegation credentials in distributed storage is painful and time-consuming. One may argue that the communication cost may be reduced by caching frequent access delegation chains and shortening long chains to short ones [12]. However, if the credential pool is large, these fixes may not help much.

We propose an alternative mechanism for delegation of authority in decentralized trust management systems that has several advantages over existing role-based delegation method. We give the notion of delegation by building *top-down* delegation chain. Top-down delegation is privilege-oriented, in which one can only delegate a right to others only if he or she is delegated it. Delegations in a single delegation chain happen sequentially. In almost all the existing delegation models, individual delegations, in a form that Bob delegates to Alice, are independently made and stored, and then delegation discovery algorithms sort them out among a large credential pool and piece them into a delegation chain dynamically [12, 33]. We call them individual delegation approach. The time when these individual delegations are issued may not in the same order as it in the final delegation chain. The delegation credentials in this case are mostly distributed and the process to gather relevant delegations and build delegation chain from distributed storage is non-trivial [12, 33, 30]. This is because the distributed storage of potentially large number of single delegation credential.

Top-down delegation makes delegation chain discovery straightforward. Delegation chain information is on a delegation credential. Each delegation is equivalent to extending the existing delegation chain by introducing a new node. All the existing nodes are the ancestor nodes of the newly added one. This implies that the information of all the entities involved in the delegation chain are on the delegation credential. The information of the entities is ordered the same order as they are delegated. So complicated delegation chain discovery algorithms [12, 33] are virtually eliminated. One should not confuse our right-oriented delegation with capability-list style delegations such as KeyNote [7]. Supporting role-based delegation, rights in our model have broader and more abstract sense. A right may be a role assignment, as well as an action. This way one can map roles in different organizations in one delegation. Top-down delegation approach does not significantly increase total number of delegation credentials.

Top-down delegation method makes more precise delegations than existing role-based delegation mechanism [33]. It supports privilege-based delegations and third-party delegations, and allows issuers to have tighter control over the resources. In existing role-based delegation model such as *RT* framework [33], the discovered delegation chain may end up delegating privileges to ones that the delegators have no intention to. To see why this is the case, consider the following example. HospitalB has a long standing delegation policy that says lab managers at Genome lab may have all the privileges of their doctors. On a recent collaboration project between HospitalB and HospitalA, HospitalA delegates to role doctors at HospitalB the rights to read its object o. Because o has confidential information about the collaboration, HospitalB may not want to let lab managers at Genome lab to access it. It may wants to delegate this privilege to others who it thinks are appropriate. In top-down delegation model, a lab manager is unable to access o because it does not have a delegation credential that specifically delegates him this privilege. On the contrary, a delegation model such at *RT* framework will allow a lab manager at Genome lab to access o. In that, the delegation credential discovery algorithm first locates both individual delegations. Since lab managers at Genome lab can have the same privileges as doctors at HospitalA, who are delegated to access o, the result of the chain discovery algorithm will incorrectly allow lab managers at Genome lab to access o. This is contrary to the intention of HospitalB. The security of resources is seriously compromised.

We present *RBDD*, a decentralized role-based decentralized delegation model that is based on Hierarchical Identity-Based Encryption (HIBE) scheme. HIBE scheme makes implementation of top-down delegation easy, although the top-down delegation mechanism does not depend on HIBE scheme. In our model, the delegation of authority is represented as issuing a public/private key pair in HIBE scheme. The identity-based encryption technique allows the use of any string as public keys. In RBDD, we utilize this feature to simplify the delegation authentication process and save communication costs. A delegation public key is a human readable string that contains information of a delegation chain, which includes the identity of the original issuer and delegated entities, and delegated privileges. It may also include delegation constraints such as expiration date, etc. The process of delegation authentication eliminates the need of dynamic delegation chain discovery. Verifying a delegation credential is to verify whether the user has the private key corresponding to a delegation public key. The authorization process is simplified as well. The delegation chain information is stored in user's delegation public key, which is used at authorization to determine the validity of the delegation. In a decentralized trust management setting that involves a large number of participants from multi-organizations, these simplifications largely reduce the cost of delegation chain discovery.

We introduced the concept of top-down delegation approach and explained why it can solve distributed delegation chain discovery problem. The rest of the paper is organized as follows. In Section 2 we discuss existing decentralized trust management models and related techniques; in Section 4 an example scenario is presented where RBDD can be applied; in Section 3 we discuss some of the preliminary knowledge of our design, including Identity-Based and Hierarchical Identity-based Encryption schemes, role-based access control protocols. Then we divide the discussion of our design of RBDD model in the order as follows. A basic RBDD model is described in Section 5, including an overview in Section 5.1, how to initiate a delegation chain in Section 5.2, how to extend a delegation chain in Section 5.3, and authentication in Section 5.4. In Section 5.5, we give practical solutions to the credential storage problem, in case when the number of receivers of a delegation is large. We make several extensions to our basic RBDD model in Section 6, which makes

the model more expressive and efficient. In Section 7, delegation revocation and security are discussed in Section 7.1 and Section 7.3, respectively. We also discuss the efficiency and usability of top-down delegation method based on HIBE scheme, in comparison with existing delegation approaches. Last, Section 8 is the conclusion and discussion of future work.

2 Related Work

A similar decentralized access control model to what we present here is *RT* framework, a family of Role-based Trust management languages for representing policies and credentials in decentralized authorization [32]. *RT* focuses on the Trust-management language aspects of attribute-based access control, where access control decisions are based on authenticated attributes of the subjects, and attribute authority is decentralized. *RT* uses the roles to represent attributes. The delegation chain in *RT* framework is built on dynamically piecing together individual delegation credentials among a large credential pool.

The policyMaker [8] and KeyNote [7, 6] trust management systems present a very generalized approach to specifying and interpreting security policies, credentials and trust relationships. The system puts all the policy and credential information into signed certificates that can be stored in a distributed fashion. Certificates in PolicyMaker are generalized as they consist of programs written in a general programming language. Since an access policy is represented by the set of all such certificates, it may have low readability for a resource owner. KeyNote is a capability-style system and settles on a C-like expression and regular expression syntax for describing conditions. The policy for a resource use is implied by the combination of all the assertions.

SPKI/SDSI (Simple Public Key Infrastructure/Simple Distributed Security Infrastructure) is a public-key infrastructure emphasizing naming, groups, ease-of-use, and flexible authorization [12, 1, 17, 3]. SPKI proposed by IETF is a standard for authorization certificates, name certificates and access control lists that are all represented by a formalized keyword, value syntax called S-expressions. An authorization certificate consists of an issuer, and a subject, both represented by a public key; the actions that are authorized; and an optional set of validity conditions.

SDSI proposed by Lampson and Rivest is a public key infrastructure that features decentralized name space [38]. In SDSI, the owner of each public key can create a local name space relative to that key. These name spaces can be linked together in a flexible manner to enable chains of authorization and define groups of authorized principals. In 1997, the SPKI and SDSI efforts were merged, and the resulting synthesis has been called SPKI/SDSI. To access a protected resource, a client must show a proof that takes the form of a "certificate chain" proving that the client's public key is one of the groups on the resource's ACL, or that the client's public key has been delegated authority from a key in one of the groups on the resource's ACL. Due to the flexible naming and delegation capabilities of SPKI/SDSI certificates, finding such a chain can be nontrivial. The worst-case bound on its running time is polynomial in the length of its input [12].

As noted in [32] KeyNote and SPKI/SDSI are capability-style systems that are not efficient in a decentralized environment. RBDD is a role-based access control and solves the problem. Another problem with SPKI/SDSI does not discuss how to do certificate chain discovery when certificates are distributed. It assumes that all of the certificates are available to the discovery algorithms.

A cryptographic hierarchical delegation protocol was described by Ding and Petersen [16], which is based on Schnorr signature scheme [41], self-certified public keys [24], and the concept of hierarchical key generation [21]. The scheme aims at how to build the cascaded delegation chain where all principals in the chain are traceable from the delegation credential. Their scheme is based on exponentiations, which is slow compared to elliptic curve computation in BF-IBE scheme [9].

dRBAC is a decentralized trust-management and access-control mechanism for systems that span multiple administrative domains [19]. dRBAC utilizes PKI identities to define trust domains, roles to define controlled activities, and role delegation across domains to represent permissions to these activities. dRBAC supports such features as third-party delegation of roles from outside a domain's names-pace, modulation of transferred permissions using scalar valued attributes associated with roles, and continuous monitoring of

trust-relationship. The efficiency of delegation chain discovery, however, is a concern because the the number of potential authorizing paths in a delegation tree with a constant branching factor, is clearly exponential in depth. As observed by others, a significant reduction in the number of paths that must be considered is possible if the search is simultaneously conducted in both directions [19].

Cohen *et al.* described a family of coalition-based access control (CBAC) model [13] to facilitate data sharing and protect the safety of resources in dynamic coalition. Shibboleth [43] is a cross-institutional authentication and authorization service for access control to web-accessed resources. The main feature is to maintain the anonymity of user at the resource side by asking the user's home institution for attributes for the user. The user needs only authenticate to his host site, and each institution must trust all the sites at which any of its user's reside.

Permission-based delegation model (PBDM) built on RBAC supports user-to-user, role-to-role delegations [49]. A delegator creates one or more temporary delegation roles and assigns delegatees to particular roles. However, delegations in PBDM requires changes of role hierarchies by the proper authority, for example, a project leader who has write access to the role assignment and access policies. PBDM does not address how decentralized delegations, where central authorities may not have to be involved.

3 Preliminaries

In the subsequent sections, we first give definitions and notations, some of which is similar to *RT* framework [32]. Then we briefly introduce Identity-based Encryption (IBE) and Hierarchical Identity-based Encryption (HIBE) schemes.

We refer *entities*, which are also called principals, as the organization or an individual. An entity may issue credentials and make requests. One can determine which entity issue a credential or request based on the public/private key pairs. A *delegation chain* is a path that connects original delegator to an entity via other intermediate delegated entities. It is made of multiple delegations, each having a delegator and a delegatee. An *edge* on a delegation chain represents a delegation that connects two adjacent entities on the delegation chain. One of them is a delegator and the other is the delegatee of the delegation.

Role-based access control (RBAC) emerged rapidly in the 1990s as a proven technology for managing and enforcing security in large-scale enterprise-wise systems [40, 36, 48, 11, 18, 5, 39]. The central notion of RBAC is that permissions are associated with roles, and users are assigned to appropriate roles. This greatly simplifies management of permissions.

A *role* r in entity A is denoted as $A.r$. For example, HospitalA.doctor represents the role doctor in HospitalA. A role defines a group of entities who are members of this role. We also define an object o in A as $A.o$. A delegation involves three major components, delegator, delegated entity and the delegated rights. We express the delegation of the rights of reading object o in A from role r_1 in A to role r_2 in B as $A.r_1(\text{read}, A.o) \rightarrow B.r_2$. Unlike *RT* framework, delegated rights, for example (read, $A.o$), is a component of delegations in our model. It is written as the subscript of the $\rightarrow$ as $\rightarrow_{(\text{read},A.o)}$. The entity to the left of the $\rightarrow$, $A.r_1$, is the delegator, the issuer of this delegation. The entity to the right, $B.r_2$, is the receiver of the delegation. The owner of object o may not have to be the delegator. The delegated privilege can be role assignment, such as role doctor assignment.

3.1 IBE and HIBE schemes

Boneh and Franklin developed a fully functional Identity-Based Encryption scheme (BF-IBE) [9] in 2001 based on bilinear maps between groups. The notion of identity-based encryption was proposed by Shamir in 1986 [42] , whose original motivation was to simplify certificate management in e-mail system. When Alice sends mail to Bob at bob@company.com she simply encrypts her message using the public key string "bob@company.com". The most important feature is that there is no need for Alice to obtain Bob's public key certificate. In this paper, we call the public key *self-authenticated*. When Bob receives the encrypted mail he contacts a third party, which is called the Private Key Generator(PKG). Bob authenticates himself to

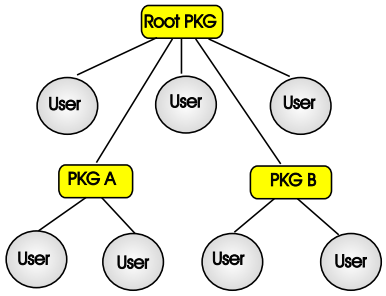

Figure 1: Root and lower-level PKGs and users in HIBE.

the PKG in the same way he would authenticate himself to a CA and obtains his private key from the PKG. Bob can then read the e-mail. Unlike the existing secure e-mail infrastructure, Alice can send encrypted mail to Bob even if Bob has not yet setup his public key certificate. Also note that key escrow is inherent in identity-based e-mail systems: the PKG knows Bob's private key. Key revocation is discussed in Boneh's paper, as well as several new applications of IBE.

Some efforts are made applying IBE scheme for secure email and access control [20, 44, 45, 31]. Smart presents a mechanism to encrypt to an arbitrary collection of identities using a variant of the Boneh-Franklin IBE scheme [44]. Smetters and Durfee deployed Boneh and Franklin IBE techniques and presented a system for secure email and IP-based network traffic [45].

A collusion resistant hierarchical identity-based encryption scheme based on [9] is presented by Gentry and Silverberg [22], which supports hierarchical organization of user's IDs and has lower-level PKGs as well as a root PKG. Our RBDD model is based on it. See Figure 1.

HIBE scheme [22, 28] reduces the burden of the root PKG for generating and updating private keys. Lower-level PKG receives its private key from its parent node in the hierarchy and it can generate private keys for its child nodes without consulting the root PKG. Lower-level PKG does not require any public parameters. Even though in 1 users and lower-level PKGs are distinguished, a user has the ability of serving as a lower-level PKG and generates private keys for his or her lower-level entities (users or lower-level PKGs) whose IDs have the user's ID as their prefixes. Delegation in RBDD utilizes this fact. The encryption/decryption time and the length of ciphertext in the HIBE scheme by Gentry and Silverberg [22] are proportional to the depth of an entity in the delegation chain. The complexity can be improved using aggregation technique similar to aggregate signature [10, 23]. The improved HIBE [47] has the same complexity of ciphertext length and decryption time as Boneh-Frankline IBE scheme [9], with the encryption complexity unchanged. This HIBE scheme requires lower-level PKGs to have their own public parameters which are used for encryption.

4 Example scenario

The protection of sensitive medical data stored in medical information system from unauthorized access or modification is a major concern in health-care collaborations. It is important to effectively share resources while keeping the confidentiality, integrity of the medical documents. In this section we describe an application scenario for RBDD. Hospitals and clinic labs maintain a large amount of medical information, including patients medical records, appointment and operation schedules, payment information, etc. Because of the sensitive nature of the medical information, access discretion is required. The administration of hospital employees is role-based. Employees are assigned to roles according to their qualification, for example, doctor, nurse practitioner, physician assistant, etc. Different roles are given different access rights or privileges to the hospital's medical information. Some roles inherit the rights of other roles. For example, doctors have all the rights that nurse practitioners have. The inheritance relationships among roles forms a role

hierarchy [40]. The access rights for resources are defined in access policies based on role assignments by the administrators.

During a collaboration project between the women's clinics, WClinics for short, at HospitalA with the neurology division of HospitalB. The participating medical workers at HospitalB need to access medical information, say object o, that belongs to HospitalA. The staff at neurology division of HospitalB delegates testing work to an independent Genome lab. The testing work requires to access o. The Genome lab decides that doctors who work in the lab can have access to the object. Doctors may then delegate this right to their assistants. Alice is a doctor assistant in Genome Lab. She has a credential issued by Genome lab to prove this role assignment. The above described credentials are shown below, based on the notations in 3. We denote the object o at HospitalA as $A.o$.

$$\text{HospitalA.WClinics\_member} \rightarrow_{(\text{read},A.o)} \text{HospitalB.neurology\_member} \tag{1}$$

$$\text{HospitalB.neurology\_member} \rightarrow_{(\text{read},A.o)} \text{Genome\_lab.doctor} \tag{2}$$

$$\text{Genome\_lab.doctor} \rightarrow_{(\text{read},A.o)} \text{Genome\_lab.assistant} \tag{3}$$

$$\text{Genome\_lab} \rightarrow_{\text{assistant}} \text{Alice} \tag{4}$$

The delegation (2) can be read as: a member of neurology division at HospitalB allows doctors of Genome lab to read object o at HospitalB. Similarly, credential (4) means that Alice is assigned role assistant at Genome lab. An assistant at Genome lab can be proved to have read access to o by linking the first three delegations together.

$$\text{HospitalA.WClinics\_member} \rightarrow_{(\text{read},A.o)} \text{HospitalB.neurology} \rightarrow_{(\text{read},A.o)} \text{Genome\_lab.doctor} \tag{5}$$

$$\rightarrow_{(\text{read},A.o)} \text{Genome\_lab.assistant} \tag{6}$$

$$\tag{7}$$

Like in *RT* framework, we assume that the delegation credentials may be stored by the issuer or by the receiver. Because individual delegation credentials ((1) to (4)) are stored distributedly across the internet, the process of finding such a delegation chain is non-trivial in existing delegation models [32]. We illustrate it below.

Upon the request of Alice to access o, HospitalA has to find out whether Alice has been delegated by HospitalA or some intermediate entities the proper rights. By looking at its delegation credential server, HospitalA finds that ten roles at other institutions are delegated the right to access o. HospitalA asks if Alice belongs to any of the delegated roles. If Alice does, the search for delegation chain stops. Otherwise, HospitalA has to query the delegation credential servers of these institutions for what roles are further delegated the rights to. Once the queries are returned, HospitalA again asks if Alice is any one of the delegated roles. The search process stops once a valid delegation chain forms between HospitalA and Alice. This search may be initiated by Alice or both HospitalA and Alice. Details of the search algorithms in *RT* framework are described by Li, Winsborough and Mitchell [33].

Even if there is a central storage for all the delegation credentials, the efficiency of delegation chain discovery is not high because of the potentially large number of paths from the original issuer to the last delegated entity [12], from HospitalA to Alice in our example.

We try to avoid the delegation chain discovery by building the delegation chains top-down. In our model, a delegation credential contains a delegation chain that has been built so far since the privilege is authorized by the original issuer. We call the first entity on the delegation chain the original issuer of a delegation. The rights may be further delegated by extending the existing delegation credential, which is similar to adding a new node in the delegation chain. A delegation credential therefore contains information of all the entities that are involved in this delegation of rights and there is no need for delegation chain discovery.

In our example, HospitalA gives the delegation (1) to HospitalB, which is stored in local server of HospitalB. Instead of just giving Genome lab delegation (2), HospitalB links (1) and (2) together, and issues Genome lab a credential shown below.

$$(\text{HospitalA.WClinics\_member} \rightarrow_{(\text{read},A.o)} \text{HospitalB.neurology\_member} \rightarrow_{(\text{read},A.o)} \text{Genome\_lab.doctor} \tag{8}$$

The linked credential is different from the *linked role* in *RT* framework [32], because only the action (read, $A.o$) is delegated, not the roles of HospitalA.WClinics_member or HospitalB.neurology_member. The linked delegation credential indicates that the chain of delegations. The doctor of Genome lab receives the delegation credential from HospitalB, who obtains the right of read $A.o$ from HospitalA. Because the right is the same through out the delegation chain, we only keep the right in the first delegation issued by the original delegator. We rewritten (8) below.

$$(\text{HospitalA.WClinics\_member} \rightarrow_{(\text{read},A.o)} \text{HospitalB.neurology\_member} \rightarrow \text{Genome\_lab.doctor} \tag{9}$$

Delegation credential (9) together with the proof of role doctor are needed in order for a doctor at Genome lab to read o at HospitalA.

There is an alternative mechanism for this delegation. Suppose that there is a large number of doctors in Genome lab and it is inefficient to issue every doctor a credential shown above. The alternative approach is to store a copy of credential (9) to doctors at the credential servers of Genome lab.

There are two cases of how the credential is retrieved. It may be retrieved by the resource owner HospitalA when a doctor Bob from HospitalB requests to access o. Bob at Genome lab does not have the delegation credential required at authorization, but shows to HospitalA the proof of his doctor identity. HospitalA contacts the credential server of Genome lab to ask about relevant delegation credential (9) about the role doctor. Once (9) is retreived, it is combined with the role credential of Bob to prove authorization. Or a client-pull architecture is used and the delegation credential can be retrieved by a doctor at Genome lab. Bob submits his request of accessing o. HospitalA replies that it does not know Bob and suggests him to contact the credential server of Genome lab. Bob queries the credential server and retrieves the delegation credential. In case when the delegation to the role doctor is kept by credential server of Genome lab and Bob wants to delegate the rights to his assistant, Bob has to first query the server to obtain a copy of the delegation credential. Then he extends the delegation to his assistant using the retrieved credential.

Roles can also be delegated, as well as actions. For example, HospitalA can selectively allow some doctors at HospitalB to have provileges of the role doctor at HospitalA. Or a professor at Computer Science Department may be delegated to have the role professor at the Math Department in a university.

Role inheritance can be expressed as delegation in our model, which is similar to *RT* framework. For example, a role lab manager inherits the privileges of role doctor at Genome lab, which is expressed as the following.

$$\text{Genome\_lab} \rightarrow_{\text{doctor}} \text{Genome\_lab.lab\_manager} \tag{10}$$

Suppose John is the lab manager at Genome lab and would like to access o. HospitalA suggests him that he may have to contact the delegation server of Genome lab for proof. John authenticates himself as a lab manager to the server and requests for any proofs for him to access o at HospitalA. The server retrieves the delegation credential, which is (9). It also gives John (10) which shows that lab manager naturally inherits the privileges of doctor. Because the role inheritance is defined as a local policy or credential, it is straightforward to locate it. By showing HospitalA (9), (10), and his lab manager credential (11), John can access o.

$$\text{Genome\_lab} \rightarrow_{\text{lab\_manager}} \text{John} \tag{11}$$

5 RBDD: Role-based Decentralized Delegation Model

The delegation credentials shown in the example are RBDD (Role-based Decentralized Delegation) credentials. We now formally describe RBDD model. We divide the discussion into the following major parts: overview, definitions of delegation public/private key, delegation public key authentication, revocation and update.

5.1 Overview

RBDD is a decentralized role-based delegation model that emphasizes on efficient delegation chain discovery by utilizing hierarchical identity-based encryption scheme. To introduce how delegation is accomplished in RBDD, we introduce four components: original delegator or issuer, intermediate delegators (or entities), the last delegated entity, resource owner. The original issuer may be a resource owner, or an entity who is authorized to delegate certain privileges as in third-party delegation. The delegation credential is in form of a public/private key pair. The credential may be stored by the issuer or by the delegated entity or both. Usually if the number of delegated entities are small, the delegation credentials can be given to each one of them. If the numbers of the recipients are large, the delegation then made to roots or administrators of the organizations where the recipients belong. The roots keep the credentials and will release a copy of it upon request. A delegated principal, say BoB, may further delegate the rights to others, say Alice, using his delegation credential. If he already has his credential, he creates a new delegation credential for Alice by extending his. The new credential corresponds to an extended delegation chain. In case that Bob has not already have his credential, he needs first query the roots of his affiliated organizations to obtain a copy of the credential. When Bob wants to access his privilege, he presents his delegation and other credentials to the resource owner, who authenticates Bob's identity and the delegation credential. Again, Bob may have to first obtain the proper delegation credential from the local credential server of his home organization.

We discuss a basic RBDD model below. In Section 6, a few extensions and optimizations are given to the basic model, such as the delegation credential may contain expiration date and other variations that make the scheme more expressive and efficient. In basic RBDD model, we assume that delegations are permanent and have no expiration date. Another assumption is that one delegates whatever rights he is entitled to. We allow third-party delegation in basic RBDD model, where an entity may delegation a right to others even though he is not entitled to exercise it. In RBDD, we assume that each entity has a public/private key pair issued by proper authorities in any PKI. This public key is used in RBDD for authentication and secure email purposes and are assumed to be public available. We divide the delegation protocol in RBDD into three parts: initiate a delegation chain, extend a delegation chain, and verify a delegation credential.

5.2 Initiate a delegation chain

In RBDD delegations are represented by public and private key pairs. To distinguish them from public/private key pairs for authentication purpose in conventional PKI, we call them delegation public and private key pairs. A delegation public key is a human readable string that describes the original issuer of the delegation, the delegated rights, and all the entities on the delegation chain. It is of form (ID_1, ..., ID_n) where $n \geq 1$ and comma is a delimiter. Each ID is a string and represents a delegation on a delegation chain. It corresponds to the identity of a node on the hierarchy of HIBE scheme. In basic RBDD, each ID string contains three fields that describe a delegation: *delegator, rights*, and *delegatee*. The delegator and delegatee may be identified by their identities: roles, names, public keys, etc. For example, HospitalB.neurology_member [1]. A delegated right may be a role or an action. The field values are concatenated into a string ID. The format of a delegation public key is shown in 2.

The original delegator initiates a delegation chain by creating a delegation to the first intermediate entity. This process can be done in three steps. First it defines a string, which includes its role or name, the rights to be delegated, and the role or name of the entity who receives the delegation. This string is

[1] The notation is explained in Section 4

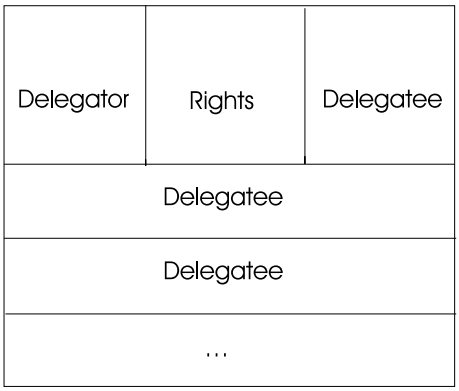

Figure 2: Delegation public key in basic RBDD model.

the *delegation public key* and represents the delegation chain. We refer this string as ID_1. For example, the string associated with (1) is (HospitalA.WClinics_member | [2] read o | HospitalB.neurology_member). Then, the original delegator runs the ROOT SETUP algorithm in HIBE scheme [22] to generate a set of public parameters, which are made public. It corresponds to the root PKG in HIBE scheme. The input identity of the *root PKG* is the string ID_1 [3]. The secret chosen in ROOT SETUP algorithm by the original delegator is the *delegation private key*. Last, the original delegator obtains the public key of the delegated entity and encrypts the secret under the public key. The encrypted secret along with the delegation public key ID_1 are sent over to the delegatee. The delegation key pair is to be used by the receiver to prove that he is properly authorized. It is also used in RBDD to generate further delegations to other entities. The original issuer keeps a copy of ID_1. The published public parameters are used for future delegations and authentication of delegation public keys. A delegation chain has been initiated. How to delegate the rights to other entities and extend the delegation are described below.

5.3 Extend a delegation chain

An entity with a valid delegation public/private key pairs may delegate the rights further to other entities whom he trusts. An extension of a delegation chain can also be made in three steps. First, the delegator parses his delegation public key as (ID_1, ..., ID_i). We denote the role or the name of the next delegated entity and other necessary information as ID_{i+1}. In the extended RBDD, ID_{i+1} may include the expiration date and other constraints to this delegation. The new delegation public key would be (ID_1, ..., ID_i, ID_{i+1}). For example, Bob with role HospitalB.neurology_member has the following delegation public key.

(HospitalA.WClinics_member | read o | HospitalB.neurology_member)

He wants to delegate the right to the role doctor at Genome lab. He defines ID_1 as string HospitalA.WClinics_member | read o | HospitalB.neurology_member, ID_2 as string HospitalB.neurology_member | read o | Genome_lab.doctor. The new delegation public key is (HospitalA.WClinics_member | read o | HospitalB.neurology_member, HospitalB.neurology_member | read o | Genome_lab.doctor).

The delegator also obtains the public parameters from the original delegator associated with this delegation chain. Second, the delegator runs the LOWER-LEVEL SETUP algorithm and EXTRACTION algorithm on input (ID_1, ..., ID_i, ID_{i+1}). It corresponds to a *lower-level PKG* in HIBE scheme. The secret generated in the EXTRACTION algorithm is the new delegation private key to be given to the delegated entity. Last, the delegator obtains the public key of the delegated entity and sends over the new delegation public key and the corresponding private key encrypted under the public key. This finishes the process of extending a

[2] We denote | as concatenation.

[3] See Section 6 to the discussion of how to reduce the number of public parameter sets.

delegation chain. The receiver uses the key pair for authorization purpose and may also delegate the rights further by running through the above process.

For some security reasons, the receiver may want to authenticate the delegator before he accepts the delegation credentials. The authentication is to ensure that the delegator is the same as the one that appears last on the delegation public key that the delegator holds. This is to prevent that an adversary steals and uses someone else's delegation key pairs to delegate. The verification process is essentially the same as the authentication process done by the resource owner. It is described below.

5.4 Authentication and authorization

In RBDD an entity has several types of credentials. One type is the public/private key pair generated in any existing public key infrastructure. This is for authentication purpose and proving identities. The other may be attribute credentials that issued by the user's affiliation to prove certain attributes such as role assignments. For example, Alice generates on her own a public/private key pair that she would use for secure email messaging. She has a public key certificate issued by a CA. Alice works in Genome lab as an assistant to doctors. Genome lab issues her a attribute certificate to prove that she or her public key has the role of doctor assistant. Those two types of credentials are not our focus in this paper and we assume they have been set up *a priori* to delegations. We are interested in the third type of credential in RDBB, which is the delegation public/private key pair. It is for the purpose of proving proper authorizations at making requests for resources out of one's role privileges. The verification of delegation key pair depends on the verification of the other types of credentials.

Because of the use of HIBE scheme [22], which is based on BF-IBE scheme [9], delegation public key is a human readable string that describes: the original issuer of the delegation, the delegated rights, intermediate entities involved, the last delegated principal at the moment. Resource owner allows access to someone it already knows and trusts. Through trust linkage of the delegation chain, it also grants access to entities who are trusted by ones the owner knows. The resource owner does not have to know all the entities involved in the delegation chain in order to make decision on authorization. It only has to trust the original delegator.

The protocol of authenticating and authorizing of an unknown entity via delegation credentials involves the following steps:

1. A user U sends a request for accessing object o owned by entity A. U also submits the delegation public key.

2. If the original delegator on the delegation public key is the resource owner A, go to step 4.

3. If the original delegator is not A, A has to find out if it is a trusted entity. It queries its policy server for the entities who have authorization to delegate access rights to o, and obtains a set of roles or names, who are entitled to access o or are authorized to delegate the right. If the original delegator is one of the trusted entities, go to 4.

4. A initiates a challenge-response to authenticate the delegation public key, which verifies U has the corresponding private key. A also verifies that the rights shown on the delegation public key is consistent with the request.If U does not correctly answers the challenge posed by A, A quits. Otherwise, A authenticates U below.

5. A verifies that U is the last delegated principal on the delegation chain. U does this by showing credentials, such as attribute certificate or public key certificate. The request is granted if the verification is successful, and denied if otherwise.

5.5 Credential storage

Readers may have noticed that one has to have his delegation credential available, in order to extend a delegation. Above we implicitly assume that the delegation credential is stored at the delegated principal

and is available when delegation is extended. However, this may not be practical when the number of delegatees are large. For example, an internet bookstore gives discount to trusted members. The privilege is delegated to the manager of a book club, via several intermediate entities. The book club wants graduate students at a state university to have the discount. However it would be unrealistic for the bookstore to issue a delegation credential to each graduate student. Instead the delegation can be stored at the credential server of the state university, a place where students know to find their credentials. The stored credential represents the delegation chain connecting from the bookstore to role graduate student at the university. When a student wants to have the discount from the bookstore and is asked for discount credential, he queries credential server of his university for it. When he wants to delegate the discount, he needs first obtain a copy of the credential from the credential server of the university too. This way, we eliminate the trouble of issuing delegation credentials to every potential subjects, and still avoid the complexity of delegation chain discovery. The abstraction of roles in RBDD greatly reduces the number of delegations [32].

6 Extensions

The delegation model described is a basic RBDD model. In this section, we introduce several extensions to the basic model that make it more expressive and efficient.

The challenge-response at verification of a delegation private key requires both parties to know the public parameters of a delegation chain, which is made public by the initial delegator. Recall the public parameters are generated by running ROOT SETUP algorithm in HIBE scheme, when a delegation chain is first initiated. In basic RBDD a set of public parameters is associated with a delegation chain. This means if HospitalA delegates the right, read o, to other two hospitals: HospitalB and HospitalC. It has to generate two sets of parameters. This certainly creates a management problem for HospitalA. For example, to initiate the delegation for HospitalB, HospitalA defines ID_1 as (HospitalA | read o | HospitalB). It runs ROOT SETUP algorithm in HIBE scheme and generates a set of public parameters. Then it does the same thing for HospitalC. It defines ID_1 as (HospitalA | read o | HospitalC). Another set of public parameters is generated. Observe that the only difference between the two ID_1 is that the name of the delegated principal. If we move the field delegated entity from ID_1 and make it ID_2 in delegation public key, the two delegations may share the same ID_1 and therefore the same public parameters.

As a matter of fact, the number of public parameters may be greatly reduced this way. Observe that in HIBE scheme only the root PKG with unique ID_1 has public parameters, while the lower-level PKGs with ID_i, where $i \geq 1$, do not have any public parameters. Each ID_1 gives rise a set of public parameters. Therefore, we can reduce the total number public parameters by making ID_1 only have general information and moving delegation specific information to ID_2. In the above example, the field delegated entity is taken out of ID_1. We go a step further. We only leave the field delegator in ID_1, make the field rights ID_2, and the field delegated entity ID_3. Future delegations introduce ID_i, each representing the delegated entities. A delegation public key is in the form of (ID_1, ID_2, ID_3, ..., ID_n), where ID_1: delegator identity, ID_2: rights to be delegated, ID_i where $i \geq 3$: delegatee identity.

This way, delegations of different rights made by the same delegator may share the same set of public parameters. Each entity E just runs ROOT SETUP algorithm in HIBE scheme with input of its name, which generates public parameters and a master secret s_1. E keeps the master secret, which will be used to generate delegation private keys. Given a right r to be delegated, E runs EXTRACTION algorithm with r and obtains a private key s_2 using s_1, which corresponds to (E, r). This private key is the secret to be used for generating delegation keys associated with right r. Given a subject S_1, E runs EXTRACTION algorithm with S_1 and obtains a private key using s_2, which corresponds to (E, r, S_1). Future delegations of r form a delegation public key of form $(E, r, S_1, \ldots, S_n)$, where S_i $i \geq 2$ are entities on the delegation chain. They do not increase the total number of public parameters.

The delegator may impose an expiration date on a delegation. HIBE scheme makes this easy because a public key can be any string. We add a field of expiration date in each ID_i in delegation public key. For example HospitalA.WClinics_member only allows HospitalB.neurology_member to read o before July

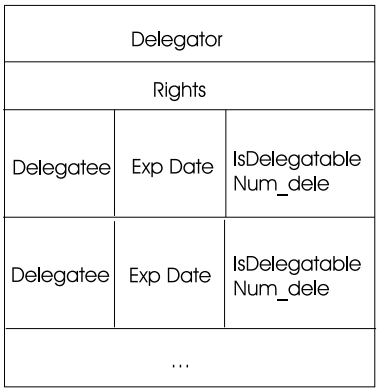

Figure 3: Delegation public key in extended RBDD model.

17th 2003. The delegation public key is (HospitalA.WClinics_member | read o before July 17th 2003 | HospitalB.neurology_member). Future delegations have to set their expiration date field no later than the expiration date of prior delegations. When authorizing access request, the verifier needs to ensure that no expiration date has been violated. In case when delegations are revoked before expiration date is reached, we bring the discuss in Section 7.1.

The delegator may also set constraints on the level of delegation, which specifies whether or not the delegation can be further delegated and for how many times [4]. By how many times we does not mean how many entities are delegated the rights, rather it indicates the length of a delegation chain. In RBDD we assume that delegations can be made to multiple entities at a time. Level of delegation information helps lower accountability problems and gives the delegator a tighter control over the delegated privileges. Two fields are introduced to each ID in the delegation public key. One indicates whether or not the rights may be further delegated, and the other indicates how many times. The verifier or the receiver of a delegation rejects a delegation credential under two circumstances. One is that either of the entities on the delegation chain is not allowed to make delegation further. The other is that numbers of delegations exceeds any of the specified value. The numbers of delegations are cumulative, and all of the specified values on the credential should be satisfied to make a delegation valid. For example, Bob makes a delegation to Alice and sets the number of delegations to one. Alice further delegates it to John and sets the number of delegations to one. John cannot make further delegations because the number that Bob sets has reached, although the number that Alice sets has not. We show the format of extended delegation public key in 3.

7 Discussion

Below we discuss various aspects of our design, including delegation revocation, security, and the complexity of top-down delegation implemented by HIBE scheme. We also compare the scalability and efficiency of the conventional delegation method with top-down delegation method.

7.1 Delegation revocation

We distinguish two cases in delegation revocation. First is delegation key revocation because the expiration date is reached. And the other is revoking delegation keys due to changes of qualifications. For example, if doctor Eve is found not qualified to read object o in HospitalA, her delegator will revoke her delegation credential regardless of its expiration date.

As the content of a public key includes its expiration date, the public key revocation in the first case is straight-forward. When a request is made, the expiration date on the delegation public key should not be

reached. The second case is slightly complicated to handle because if an intermediate entity on a delegation chain is revoked, the delegation credentials is invalid. Because the delegation chain is not dynamically discovered at verification, the resource owner has to keep track of the status of all the entities involved on a delegation chain, even though the requester is the last entity on the chain. To make things more complicated, the delegation should be revoked by any of the ancestors on a delegation path, not only the direct or the original delegator.

Revocation of delegations may be achieved by issuing short-lived delegation credentials. However, this approach causes frequent renewals and burdens the delegators. Especially the entities on the delegation chain have to coordinate their schedules, which are hard to achieve. Another approach would be one similar to revocation of public keys in any conventional PKI. Identities of revoked entities are kept in revocation servers, where others can query. In order to verify a delegation chain, one queries the server and ensures that no entity on the delegation chain has been revoked. However, decentralized delegation cannot rely on central authorities that maintain records of revoked entities, since each entity is administratively independent of each other. A reasonable approach for delegation revocation in RBDD is to let each resource owner be responsible for any revocations that concern its resources, and maintains a local revocation server. Users may be revoked by the resource owner or by any ancestor of theirs, who notifies the revocation server about the revoke decision.

Note that the delegation revocation is conceptually different from public key revocation in PKI, although connecting them together probably makes delegation revocation simple. If Eve always misuses the rights delegated to her, for example, she is careless at making delegations, then her accountability is compromised. An ancestor of hers may want to revoke the rights delegated to her. This revocation disallow Eve's ability to exercise the rights or make further delegations. Eve's own public key for secure email purposes should still be valid, so may her other delegation credentials. Revocation issues of Eve's public key in PKI can be dealt with known techniques such as authenticated dictionary techniques [26, 35, 25, 2, 27, 37] and authenticated third-party publication techniques [34, 14, 15]. We do not go into this topic in this paper.

7.2 Scalability and Efficiency

Top-down delegation mechanism completely eliminates the dynamic distributed delegation credential discovery problem. This makes authorization much faster. But what is the tradeoff? In this section we discuss the scalability and efficiency of our role-based delegation model that is built on HIBE scheme, in comparison with existing role-based delegation approach such as *RT* framework. To make the comparison more specific, we compare *RT* framework with our model below.

The reason that there is no need to dynamically discovery delegation chain is that this information is stored in delegation credentials. Therefore it is important to compare the storage requirements of both models. Two aspects are studied: the size of a delegation credential and number of overall delegation credentials. The size of each delegation credential is linear to the depth n of a delegation chain, starting from the original issuer. To be more specific, the length of a delegation public key is linear to the length of the number of entities on a delegation chain. The size of a delegation private key is constant. In *RT* framework, the credential size is constant because it only contains information of two entities, as opposed to all the ancestors on the chain.

It may seem at the first sight that our RBDD model to require higher number of delegation credentials than *RT* framework. However, RBDD model benefits from this and is able to make precise and fine-grained delegations. In order for *RT* framework to achieve the same precision in delegation, it will need the same number of credentials. Consider this situation. The role doctor at HospitalC receives a delegation from HospitalA and HospitalB, respectively. HospitalC wants to delegate the privileges to doctors at HospitalD. In our RBDD model, it needs to make two delegations to HospitalD. In *RT* framework, HospitalC just maps the role doctor to the role doctor at HospitalD in only one delegation. However, the role doctor at HospitalD receives more privileges from HospitalC than HospitalC originally intends to, which is a potential security hazard to HospitalC. Even though RBDD model requires more delegations in this example, it allows fine-grained control over the delegated resources. Only needed rights are delegated to HospitalD. To achieve

the same precision in *RT* framework, HospitalC must make specific delegations of the rights received from HospitalA and HospitalB to HospitalD [4].

In the case when HospitalC receives a large number of delegated privileges, together denoted as P, from other institutions to its role doctor. It wants to delegate P to role doctor at HospitalD. In RBDD model, instead of generating a delegation credential for each privilege in P, HospitalC may create a new role r that is given the privileges P. Then it initiates a new delegation chain and delegates the role r to the role doctor at HospitalD in one delegation. Recall HospitalC does not need to issue a delegation credential to every doctor at HospitalD. It only needs to give a copy to the credential server of HospitalD as we discussed in Section 5.5. HospitalC does not have to map its doctor role to others, if it does not want to. Therefore, with careful role-based management the number of overall delegation credentials can be greatly reduced.

There is one question remains, that is how to make the original issuers know about the new role r. Suppose a doctor at HospitalD wants to use one of the privileges in P. The resource owner is unable to recognize r, because it is a local role name at HospitalC. The mapping of role r to privileges P is defined by HosptialC, but not shown on the delegation credential. Therefore, either the requester or the resource owner has to query delegation credential servers of HospitalC to obtain this mapping, as well as the delegation credential made to HospitalC by the resource owner. The described process is a simple credential query process, that can quickly locate the required delegation credentials.

From the above discussion, we conclude that RBDD model eliminates complicated distributed delegation credential discovery and allows precise delegations. It can achieve these without a significantly increase in the overall credential storage.

7.3 Security

The security of RBDD is based on the security of HIBE scheme. Hierarchical identity-based encryption scheme has chosen-ciphertext secure under the assumption of Bilinear Diffie-Hellman Assumption (BDH) and the random oracle model [9]. The security of HIBE guarantees that private key corresponding to public key (ID_1, ..., ID_n) cannot derive the private key of ancestor nodes associated with (ID_1, ..., ID_m), where $m < n$. In RBDD this has the following implication. An entity whose identity is ID_n is given a delegation private key by ID_{n-1}. He is given the delegation private key associated with (ID_1, ..., ID_n). He cannot use this private key to compute the delegation private key of his delegator. He cannot use his private key to compute a valid delegation private key corresponding to (ID_1, ..., $\mathrm{ID'}_n$), where $\mathrm{ID}_n \neq \mathrm{ID'}_n$. So this entity cannot guess the private key of his delegator, nor can he impersonates his delegator to issue valid delegation keys to others. Authentication of delegation credential is done along with authentication of the identity of the holder. This ensures that the identity specified on the delegation credential is the one stated on the delegation public key.

In the RBDD model that we have been described so far, if a delegation key pair is compromised the adversary may use it to make delegations to anyone and compromises the security of the resources. For example, if Eve obtains a delegation from HospitalA to Alice. She can extend the delegation public key to make a delegation to her, and compute the corresponding private key from the stolen one. It is as if Alice delegated the rights to her. Note that Even cannot directly use the stolen credential because she does not have identity credentials of Alice's, which will be needed for verification. We can prevent this type of break in by requiring a signature from the delegator with each delegation. A delegation credential includes not only a delegation public/private key pair, but alo a collection of signatures, the number of which is the number of entities on the delegation chain. The signature is signed by the delegator to prove the authenticity of the delegation. The delegator forwards his signature to the delegated entity, along with the existing signatures that are given to him. The details are as follows.

At each delegation, the delegator generates the delegation public and private keys as described in Section 5. He then signs the delegation public key with his public key, which is for secure email and authentication purpose. This signature together with the signatures that are given to him by his delegator are sent to the next delegated entity. As previous, the delegator encrypts the delegation private key, and sends it to the

[4] In order for *RT* framework to do this, it will need to support third-party delegation first.

delegated. When answering a service request, the resource owner authenticates the delegation credential as described in Sectionaa. In addition, it verifies all the signatures that come along with the delegation credential. The request is granted only if all the signatures are successfully verified. This way we prevent uncontrolled delegation using a compromised delegation credential. However, in order to verify the signatures, the resource owner has to know or find out the public keys of all the entities on the delegation chain. Without the signatures, the verifier only has to know the HIBE public parameters of the original issuer and the public key of the requester.

8 Conclusion and Future Work

In this paper, we proposed an alternative mechanism for delegation of authority in decentralized trust management systems. We gave the notion of delegation by building *top-down* delegation chain. Top-down delegation is privilege-oriented, in which one can only delegate a right to others only if he or she is delegated it. Top-down delegation makes delegation chain discovery straightforward because information of all the entities involved in the delegation chain are on the delegation credential. The information of the entities is ordered the same order as they are delegated. So complicated delegation chain discovery algorithms [12, 33] are virtually eliminated. This is very important in decentralized trust management systems because delegation credentials are usually distributed and finding relevant credentials among a large credential pool is nontrivial.

Top-down delegation method also makes more precise delegations than existing role-based delegation mechanism [33]. It supports privilege-based delegations and third-party delegations, and allows issuers to have tighter control over the resources. In comparison, existing role-based decentralized delegation models end up delegating privileges to ones that the delegators have no intention to.

We then presented our Role-based Decentralized Delegation (RBDD) model, which is based on Hierarchical Identity-Based Encryption (HIBE) scheme. Although the top-down delegation mechanism does not depend on HIBE scheme, the novel use of HIBE scheme in delegation makes it straightforward. We discussed our design of RBDD, focusing on definitions and formalization of delegation credentials and delegation verification. The efficiency and scalability of role-based top-down delegation and the conventional delegation approach are compared. We concluded that role-based top-down delegation based on HIBE scheme eliminates the delegation chain discovery problem, without significant increasing the overall number of delegation credentials. Therefore RBDD model is a promising new decentralized delegation model.

Future plan on RBDD includes implementation of RBDD in Java and tests. We plan to use Extensible Access Control Markup Language (XACML) [46] an policy language prototype in our model. It is XML specification for expressing policies for information access over the Internet. Sun Microsystem recently released an implementation of XACML in Java [29]. The current model does not support delegation of authority. So we plan to extend the language to support delegations first and use it to express delegation policies in RBDD. As we mentioned in Section 7.1, we would like to look into the possibilities of applying Certificate-based Encryption scheme to RBDD, which may solve the delegation revocation issues with the help of CAs. We would also want to extend the model with hierarchical resources. The current resources in RBDD are discrete objects. Introducing hierarchical resources may complicate the authorization because of the need to inference privileges. Another direction to extend the delegation model is to include attributes and parameters, which we didn't have the time to put these into the current model.

References

[1] Ó. Cánovas and A. F. Gómez. A distributed credential management system for spki-based delegation systems, 2002. `http://www.cs.dartmouth.edu/~pki02/Canovas/paper.pdf`.

[2] A. Anagnostopoulos, M. T. Goodrich, and R. Tamassia. Persistent authenticated dictionaries and their applicaitons. In *Proceedings of Information Security Conference – ISC 2001*, volume 2200 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[3] T. Aura. Distributed access-right management with delegation certificates. In *Secure Internet Programming*, 1999.

[4] E. Barka and R. Sandhu. Framework for role-based delegation models. In *16th Annual Computer Security Applications Conference (ACSAC'00)*, December 2000.

[5] J. F. Barkley, A. V. Cincotta, D. Ferraiolo, S. Gavrilla, and D. R. Kuhn. Role based access control for the world wide web. In *20th National Computer Security Conference*, 1997.

[6] M. Blaze, J. Feigenbaum, and J. Ioannidis. The keynote trust-management system. Version 2. `http://www.cis.upenn.edu/~keynote/Papers/rfc2704.txt`.

[7] M. Blaze, J. Feigenbaum, and A. D. Keromytis. Keynote: Trust management for public-key infrastructures. In *Proceedings of Security Protocols International Workshop*, 1998.

[8] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of IEEE Conf. on Privacy and Security*, 1996.

[9] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*. Springer, 2001.

[10] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT 2003*, pages 416–432, 2003.

[11] R. A. Botha and J. H.P. Eloff. Designing role hierarchies for access control in workflow systems. In *25th Annual International Computer Software and Applications Conference – COMPSAC'01*, 2001.

[12] D. Clarke, J-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certificate chain discovery in spki/sdsi. *Journal of Computer Security*, 9(4), 2001.

[13] Eve Cohen, Roshan K. Thomas, William H. Winsborough, and Deborah Shands. Models for coalition-based access control (cbac). In *SACMAT 2002*, pages 97–106, 2002.

[14] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. G. Stubblebine. Flexible authentication of xml documents. In *ACM Conference on Computer and Communications Security*, 2001.

[15] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine. Authentic third-party data publication. In *DBSec*, 2000.

[16] Yun Ding, Patrick Horster, and Holger Petersen. A new approach for delegation using hierarchical delegation tokens. In *2nd Int. Conference on Computer and Communications Security*, pages 128 – 143. Chapman and Hall, 1996.

[17] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Yloenen. Simple public key certificate. `http://www.ietf.org/rfc/rfc2693.txt`.

[18] D. Ferraiolo and R. Kuhn. Role-based access control. In *15th National Computer Security Conference*, 1992.

[19] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti. drbac: Distributed role-based access control for dynamic coalition environments. In *ICDCS 2002*, 2002.

[20] Theodoulos Garefalakis and Chris J. Mitchell. Securing personal area networks. In *13 th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications - PIMRC*, 2002.

[21] C. Günther. An identity-based key exchange protocol. In *Advances in Cryptology - Eurocrypt '89*, volume 434 of *LNCS*, pages 29–37. Springer-Verlag, 1989.

[22] C. Gentry and A. Silverberg. Hierarchical id-based cryptography. In *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*. Springer, 2002.

[23] Craig Gentry. Certificate-based encryption and the certificate revocation problem. In *EUROCRYPT 2003*, pages 272–293, 2003.

[24] M. Girault. Self-certified public keys. In *Advances in cryptology - Eurocrypt '91*, volume 547 of *Lecture Notes in Computer Science*, pages 490–497. Springer, 1991.

[25] M. T. Goodrich, R. Tamassia, and J. Hasić. An efficient dynamic and distributed cryptographic accumulator. In *Proceedings of Information Security Conference – ISC 2002*, volume 2433 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[26] M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *Proceedings of DARPA Information Survivability Conference and Exposition – DISCEX '01*, volume 2. IEEE Press, 2001.

[27] M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen. Authenticated data structures for graph and geometric searching. Technical report, Center for Geometric Computing, Brown University, 2002.

[28] J. Horwitz and B. Lynn. Toward hierarchical identity-based encryption. In *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*. Springer, 2002.

[29] Sun's XACML Implementation. `http://sunxacml.sourceforge.net/`.

[30] Seungjoon Lee, Rob Sherwood, and Bobby Bhattacharjee. Cooperative peer groups in nice. In *IEEE Infocom 2003*, April 2003.

[31] I. Levy. Identifier based pkc - potential applications. In *1st Annual PKI Research Workshop*, 2002. `http://www.cs.dartmouth.edu/~pki02/Levy/slides.pdf`.

[32] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *IEEE Symposium on Security and Privacy*, 2002.

[33] 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.

[34] C. Martel, G. Nuckolls, M. Gertz, P. Devanbu, A. Kwong, and S. G. Stubblebine. A general model for authentic data publication. In *UC Davis Student Workshop on Computing*, 2001.

[35] M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proceedings of 7th USENIX Security Symposium*, 1998.

[36] J. S. Park, R. Sandhu, and G.-J. Ahn. Rbac on the web. In *ACM Transactions on Information and Systems Security*, volume 4(1), 2001.

[37] D. J. Polivy and R. Tamassia. Authenticating distributed data using web services and xml signatures. In *ACM Workshop on XML Security*. ACM Press, 2002.

[38] R. L. Rivest and B. Lampson. Sdsi - a simple distributed security infrastructure. `http://theory.lcs.mit.edu/~rivest/sdsi10.ps`.

[39] R. Sandhu, D. Ferraiolo, and R. Kuhn. The nist model for role-based access control towards a unified standard. In *ACM Workshop on Role-Based Access Control*, 2000.

[40] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29, Number 2, 1996.

[41] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991.

[42] A. Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology – Crypto '84*, volume 196 of *Lecture Notes in Computer Science*. Springer, 1984.

[43] shibboleth. `http://middleware.internet2.edu/shibboleth/`.

[44] Nigel P. Smart. Access control using pairing based cryptography. In *CT-RSA*, LNCS, pages 111–121. Springer-Verlag, 2003.

[45] D.K. Smetters and Glenn Durfee. Domain-based administration of identity-based cryptosystems for secure email and ipsec. In *12th USENIX Security Symposium*, 2003.

[46] XACML. `http://www.oasis-open.org/committees/xacml/`.

[47] Danfeng Yao. An efficient hierarchical identity-based encryption scheme. `http://www.cs.brown.edu/people/dyao/short-cbe.pdf`.

[48] N. Yialelis, E. Lupu, and M. Sloman. Role-based security for distributed object systems. In *5th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises – WET ICE'96*, 1996.

[49] Xinwen Zhang, Sejong Oh, and Ravi Sandhu. Pbdm: A flexible delegation model in rbac. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 149 – 157. ACM Press, 2003.