Role-based Cascaded Delegation: Model and Implementation

Danfeng Yao[*] Roberto Tamassia[*] William H. Winsborough[††]

Abstract

We propose *role-based cascaded delegation*, a model for delegation of authority in decentralized trust management systems. We show that role-based cascaded delegation combines the advantages of role-based trust management with those of cascaded delegation. We also present an efficient and scalable implementation of role-based cascaded delegation using Hierarchical Certificate-Based Encryption (HCBE), where the authentication information for an arbitrarily long role-based delegation chain is captured by one short signature of constant size. This implementation also provides strong privacy protection for delegation participants.

1 Introduction

Decentralized trust management (TM) systems are access control systems that allow initially unknown entities from different administrative domains to interact and establish trust with each other through mutually trusted entities. Several trust management systems have been proposed in recent years to address authorization issues in decentralized environments, e.g., PolicyMaker [8], KeyNote [6, 7], SPKI/SDSI [1, 13], and the *RT* framework [20].

The notion of delegation is essential in transferring trust and authorization in TM systems. Delegation chains connect entities trusted by the resource owner with unknown users, and play a major role in decentralized authorization. Namely, discovering and verifying delegation chains are key issues in trust management.

The problem of *delegation chain discovery* (also called credential chain discovery) [21] consists of determining whether a delegation a chain exists between two entities and, if so, finding it. The *credential chain verification* problem is to verify the discovered credentials associated with a delegation chain. In the next two sections, we discuss these problems in more details.

1.1 Delegation chain discovery

Most of the existing work addressing the discovery problem [1, 6, 7, 8, 13] assumes that all the potentially relevant credentials are available in one central storage. Li *et al.* [21] are the first to present goal-directed credential chain discovery algorithms for distributed storage. The algorithms dynamically search for relevant credentials from remote directories to build a *proof graph*.

A flexible delegation model is given in *RT* [20]. However, its delegation chain discovery algorithms may incur heavy communication overhead between delegation participants in distributed settings. For example,

[*]Computer Science Department, Brown University, Box 1910, Providence, RI, 02912, {dyao,rt}@cs.brown.edu

[†]Center for Secure Information Systems, Mail Stop 4A4, George Mason University, Fairfax, VA, 22030-4444, wwinsbor@gmu.edu

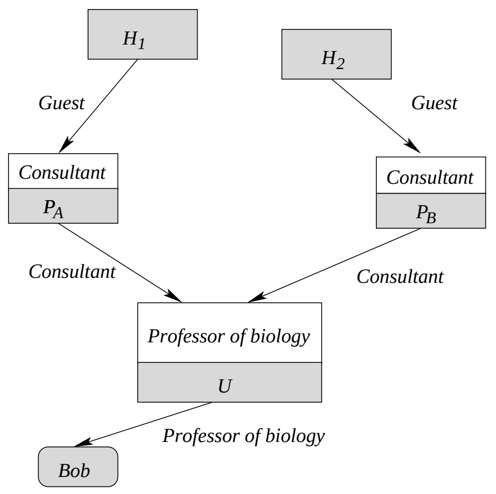

Figure 1: A schematic drawing of the delegation relationships described in Section 1.1. Arrows represent the direction of the delegation action. Dotted lines represent other possible delegations.

Bob at university U is a *professor of biology*, and he has a corresponding *role credential* issued by U. The role *professor of biology* at U is delegated the role *consultant* by two pharmaceutical companies P_A and P_B, respectively. Each of them issues the university U a *delegation credential*, which is kept on U's credential server. The role *consultant* of company P_A is delegated the role *guest* by the hospital H_1. Similarly, *consultant* of company P_B is delegated role *guest* by hospital H_2. Companies P_A and P_B keep the delegation credentials on their servers, respectively. The delegation relationships are shown in Figure 1.

In order for Bob to use his delegated role *guest* of H_1, he needs to discover all the roles he is a member of. In the forward search algorithm [21] two messages are exchanged for each edge of the graph in Figure 1, one for requesting credentials and the other for returning credentials. Therefore, the number of messages exchanged to discover a single delegation path is proportional to the number of edges of the entire proof graph. The potential high communication costs incurred in discovering delegation chains in delegation networks is also reported by Aura [3].

Distributed credential discovery algorithms also have the following two characteristics. First, they require entities or their responders (directories) involved on the delegation chain to be available and participate in the computation. Second, each node has to return to its parent node all the members of a role, or all the roles that it belongs to in the search algorithms. Most of the computed results contain private information and are irrelevant to the parent node and the delegation chain to be discovered.

Our proposed role-based cascaded delegation protocol introduced in Section 1.3 borrows the idea of cascaded delegation from proxy authentication [28, 29, 25, 15] and provides an alternative mechanism for the role-based delegation of privileges that avoids the distributed credential chain discovery.

1.2 Delegation chain verification

The verification of a *role-based* delegation chain may be quite complex in the existing decentralized role-based delegation models. Members of a role may delegate the role to others. The delegation credential is issued and signed by an individual. This requires the verification of not only the delegation credentials, but also the role credentials of intermediate delegators on the delegation chain to ensure that the delegators have the required roles to make delegations. Therefore, even if an entity is neither the requester nor the verifier, it has to participate in the verification process and prove its role membership. Alternatively, an intermediate delegator can pass down its role credential to the delegated entity to avoid participation in the chain verification. However, the delegator may consider the signature on his role credential sensitive and does not want to disclose it to the delegatee.

The third-party participation requirement can be avoided in our role-based cascaded delegation model. The sensitive signature problem is solved in our implementation of role-based cascaded delegation using Hierarchical Certificate-based Encryption (HCBE) [18] scheme in Section 4.

1.3 Role-based cascaded delegation

We propose an alternative model for the delegation of authority in role-based decentralized trust management systems, called *role-based cascaded delegation*. This model combines the advantages of role-based trust management [20] with those of cascaded delegation in distributed systems [28, 29, 25, 15, 23]. The distributed cascaded delegation problem is essentially to design a delegation mechanism that efficiently verifies a hierarchical delegation chain. In such cascaded delegation mechanisms, delegation credentials are typically passed along with each stage of the cascade and digitally signed at each transition. Therefore, the delegation chain is stored in delegation credentials and does not have to be discovered. However, previous cascaded delegation protocols do not support the use of roles in the delegation, and therefore do not have the benefits of the scalability and efficiency provided by role-based access control [27, 30, 17, 26].

Our role-based cascaded delegation inherits the basic delegation mechanism of cascaded delegation, which eliminates the need for distributed delegation chain discovery. Furthermore, delegations can be issued to roles of administratively different domains and a delegator can issue delegations to a role without knowing the members of that role. A role r is delegated a privilege by receiving a delegation credential that explicitly authorizes the privilege to role r. To cascade the delegation to another role r', a member D of the role r uses his delegation credential C to generate a delegation credential C', which includes information about preceding delegations and the role membership of the delegator D. The verifier can make authorization decision based on delegation credential C' and the role credential of the requester.

The definition of delegation chain in our model is slightly different from the delegation chain in some role-based trust-management systems [21, 13]. Namely, in our model, a delegation chain is privilege-oriented, and represents the path on which a delegated privilege is transferred among roles and entities. The *length* of a delegation chain is defined as the number of delegations on the chain.

Our role-based cascaded delegation model can be used to facilitate large scale dynamic sharing of resources in decentralized pervasive collaborative environment. It is suitable for collaborative tasks where roles from administratively independent domains are dynamically joined according to the needs of the tasks.

1.4 Efficient implementation

A main concern regarding the usability of our role-based cascaded delegation is the size and number of credentials an individual has to store, or transmit for delegation and verification. In existing cascaded delegation protocols, delegation credentials are lengthy because verification of a delegation chain requires linear number signatures in the number of entities on the chain. Conventional signature schemes, such as RSA and DSA, produce relatively long signatures compared to the provided security. For a 1024-bit modulus (security level), RSA signatures are 1024 bits long and standard DSA signatures are 320 bits long. The number of signatures required to authenticate a role-based delegation chain of length n is more than n, including signatures for proving role memberships of intermediate delegators and each of the delegation operations. Among the signatures associated with a delegation chain, the signature on a role credential is generated by the administrator of that role independently from the rest of the signatures.

Unfortunately, how to aggregate individually generated signatures from different signers on different messages are not known in conventional cryptosystems, such as RSA [11]. This means that the entire set of signatures has to be stored by delegated entities, and transmitted across networks at each delegation and verification. Because intermediate delegators in our model may be individuals who have limited computational power and communication bandwidth, implementation of role-based cascaded delegation using conventional credentials is inefficient. Another potential issue in implementing role-based cascaded delegation is privacy protection. The protocol should handle the case where the delegator considers signatures on credentials sensitive and does not want to disclose them to the delegatee. Traditional credential systems cannot provide this privacy protection.

We overcome these problems by implementing the role-based cascaded delegation using the Hierarchical Certificate-based Encryption (HCBE) [18] scheme. Using HCBE allows that the authentication information of the entire role-based delegation chain is captured by *one* short signature of constant size (about 170 bits), which makes the role-based cascaded delegation practical and efficient. This is possible by the underlining short signature [12] and aggregate signature [10] techniques. Sensitive signatures are also protected in this implementation, because individual signature, once aggregated, can be verified without being disclosed.

One may argue that users do not have to carry any of the credentials, as long as they know where to find them by keeping a reference rather than the credentials themselves. However, this approach essentially boils down to a centralized credential storage, which is not feasible for decentralized trust management [21].

1.5 Our contributions

In this paper we formalize the role-based cascaded delegation model, which combines the role-based trust management and the cascaded delegation. Role-based cascaded delegation model supports flexible and scalable decentralized role-based delegations, where delegations may be issued without the participation of the administrator of a domain. The use of cascaded delegation conveniently eliminates the need for dynamic distributed delegation chain discovery, and thus significantly reduces the communication and computation costs. User privacy is better protected in this model, because only minimum number of delegation credentials are revealed.

We present an implementation of role-based cascaded delegation protocol using the Hierarchical Certificate-based Encryption scheme. The novel use of HCBE scheme in role-based delegation allows the authentication information of the entire role-based delegation chain to be captured by one short signature of constant size, and makes the role-based cascaded delegation practical and efficient. This protocol also offers strong privacy protection, because individual signatures can be verified without being disclosed.

1.6 Organization of the paper

We introduced the concept of role-based cascaded delegation approach above. The rest of the paper is organized as follows. Discussions about roles and their delegation scopes are in Section 2. Descriptions about the HCBE scheme, terms, and our language model are given in Section 3. The definitions of the role-based cascaded delegation and our implementation are described in Section 4. In Section 5, delegation revocation, privacy, security, scalability, and efficiency issues are discussed. We discuss and compare some of the existing decentralized trust management models in Section 6. The detailed descriptions of algorithms in the HCBE scheme are in Appendix A.

2 Roles and their scopes

In our model, we define the administrator of a role as the organization that creates and manages the role. If a role credential of an entity D is signed and issued by the administrator of a role, that role is the *affiliated role*[1] of D. Otherwise, if the role credential is issued through delegation and signed by entities other than the administrator, the role is the *delegated role* of an entity. For example, Bob has a *professor* role credential signed by the university U, so *professor* at U is an affiliated role of Bob. Alice, whose affiliated role is not *professor* at U, is delegated the role by Bob, so the role is a delegated role of Alice.

The affiliated role and the delegated role are defined to have different access scopes. Delegations to a role r of an organization only apply to those who have r as the affiliated role. In the above example, if the role *professor* at the university U is delegated a privilege, Bob is entitled to this delegation, whereas Alice is not. She can only access resources at university U using the delegated role *professor*. This is different from the conventional delegation models, where delegations to a role *automatically* propagate to *all* the entities that are delegated the role. If a hospital H delegates the right of reading a patient's medical record to the role *professor* at U, Alice would be entitled to this privilege. However, for sensitive data such as medical records, this automatic propagation of delegations to unknown roles may not always be desired by resource owner. In particular, an attacker may use the delegation to construct a valid delegation chain, which gives him the authorization that the resource owner is not aware of [29].

To support flexible decentralized delegation, we give both types of a role the capability to delegate the role to others. Both Bob and Alice are able to delegate *professor* at U to other roles.

3 Preliminaries

In this section, we describe the Hierarchical Certificate-based Encryption (HCBE) schemes. Then terms and our language model are defined.

3.1 HCBE schemes

Hierarchical Certificate-based Encryption (HCBE) scheme [18] is essentially a public key cryptosystem, where messages are encrypted with public keys and decrypted with corresponding private keys. What is unique about HCBE is that it makes the decryption ability of a keyholder contingent on that keyholder's acquisition of a hierarchy of signatures from certificate authorities. To decrypt a message, a keyholder needs both his private key and the public key certificates (signatures) that are respectively signed by a chain of CAs. As usual, the CA hierarchy consists of a root CA and lower-level CAs. Higher-leve CA certifies the public key of the next-level CAs, and the CAs at the bottom (leaf positions) of the hierarchy certify the public keys of individual users. The HCBE scheme [18] is based on the aggregate signature scheme [10, 12], which supports aggregation of multiple signatures on distinct messages from distinct users into one short signature. The HCBE scheme [18] has six algorithms, **Setup**, **Certification_of_CA**, **Certification_of_Bob**, **Aggregation**, **Encryption**, and **Decryption**. The second and the third algorithms are essentially the same, one for

[1]This type of role is usually obtained through affiliation with an organization, and thus the name.

certifying the public keys of CAs, and the other for an individual. The detailed descriptions of the algorithms are in Appendix A.

3.2 Terminology

As the *RT* framework [21], we refer *entities* as the organization or an individual. An *entity* may issue credentials and make requests. An entity may have one or more affiliated roles or delegated roles. The roles are proved by role credentials. *Affiliated role credential* is the credential for an affiliated role, and is signed by the administrator of the role. Similarly, *delegated role credential* is the credential for proving a delegated role. A *privilege* can be a role assignment, or an action on a resource.

An *extension signature* is a single signature signed by the delegator at each delegation, for authenticating the delegation transaction. A *role signature* of an entity is the signature on an affiliated role credential of the entity. The *identity signature* of an entity is a signature computed by the entity using his private key. A *complete delegation credential* includes the identity signature of the requester, extension signatures, and role signatures. A *partial delegation credential* is the delegation credential issued to a role. It cannot be used by an individual for proving authorization, as it misses the identity and role signatures of the requester.

3.3 Language model

A role r in entity A is denoted as $A.r$. A is the administrator of the role $A.r$. A role defines a group of entities who are members of this role. If an entity D has an affiliated role $A.r$, his role credential is represented as $A \xrightarrow{A.r} D$, which is read as D is assigned the role $A.r$ by the role administrator A. Entity A delegates a role $A.r$ to a role $B_0.r$ by issuing a delegation credential, which is represented as $A \xrightarrow{A.r} B_0.r$. Any member D of the role $B_0.r$ can further delegate the role $A.r$ to the role $B_1.r$, which is represented as $D \xrightarrow{A.r} B_1.r$.

4 Role-based Cascaded Delegation Protocol

We first define the role-based cascaded delegation protocol, then describe our implementation using the HCBE scheme [18]. In order to make notations less complex, in what follows, the role r represents only affiliated role unless specified.

4.1 Definitions of the protocol

A role-based cascaded delegation protocol defines four operations: INITIATE, EXTEND, PROVE, and VERIFY.

- INITIATE(P_{D_0}, $D_0.priv$, $A_1.r_1$):
  This operation is run by the administrator D_0 of a privilege $D_0.priv$ to delegate to an affiliated role $A_1.r_1$. This operation initiates a delegation chain of the privilege $D_0.priv$. Inputs are the public key P_{D_0} of entity D_0, the delegated privilege $D_0.priv$, and the role $A_1.r_1$. The output of this operation is a partial delegation credential C_1 for the role $A_1.r_1$, represented as $D_0 \xrightarrow{D_0.priv} A_1.r_1$.
- EXTEND ($P_{D_n}, D_0.priv, C_n, R_{D_n}, A_{n+1}.r_{n+1}$):
  This operation is run by an entity D_n who is a member of an affiliated role $A_n.r_n$. D_n has a partial delegation credential C_n, and wants to further delegate the privilege $D_0.priv$ associated with C_n to another role $A_{n+1}.r_{n+1}$. The inputs are the public key P_{D_n} of the entity D_n, the privilege $D_0.priv$, the credential C_n of the role $A_n.r_n$, the role credential R_{D_n} of the entity D_n, and the role $A_{n+1}.r_{n+1}$. The credential C_n is represented as a delegation chain below, where P_{D_0} is the public key of the resource

owner D_0, and $A_i.r_i$ is the role that is delegated the privilege $D_0.priv$ by an entity D_{i-1} who has the affiliated role $A_{i-1}.r_{i-1}$, for $i \in [1, n]$.

$$\begin{array}{l}(P_{D_0} \xrightarrow{D_0.priv} A_1.r_1),\\ (A_1 \xrightarrow{A_1.r_1} P_{D_1}), (P_{D_1} \xrightarrow{D_0.priv} A_2.r_2), \ldots\\ (A_{n-1} \xrightarrow{A_{n-1}.r_{n-1}} P_{D_{n-1}}), (P_{D_{n-1}} \xrightarrow{D_0.priv} A_n.r_n)\end{array}$$

The EXTEND operation uses C_n to output a credential C_{n+1}, which is a function of the credential C_n, the role credential R_{D_n} representing $A_n \xrightarrow{A_n.r_n} P_{D_n}$, and the delegation $P_{D_n} \xrightarrow{D_0.priv} A_{n+1}.r_{n+1}$. Such a credential C_{n+1} may simply be delegation credential C_n plus the two individual credentials. Alternatively, D_n can compute a delegation credential for the role $A_{n+1}.r_{n+1}$ as in existing cascaded delegation protocols [25, 15], and also passes down his role credential to members of the role $A_{n+1}.r_{n+1}$. In comparison, our implementation using HCBE [18] scheme provides a more efficient approach.

- PROVE($P_{D_n}, D_0.priv, R_{D_n}, C_n$):
  This operation is performed by the requester D_n who wants to exercise privilege $D_0.priv$. D_n is a member of the affiliated role $A_n.r_n$. The entity D_n uses his delegation credential C_n and the affiliated role credential R_{D_n}, which represents $A_n \xrightarrow{A_n.r_n} D_n$, to prove to the verifier that he is authorized $D_0.priv$. The inputs are the public key P_{D_n} of the requester D_n, privilege $D_0.priv$, the affiliated role credential R_{D_n} of the requester, and the delegation credential C_n. The operation produces a proof F.
- VERIFY($[D_0.priv, P_{D_0}, A_1.r_1, P_{D_1}, A_2.r_2, \ldots, P_{D_{n-1}}, A_n.r_n, P_{D_n}], F$):
  This operation is performed by the verifier D_0 to verify that the proof F produced by the requester D_n correctly authenticates the delegation chain of privilege $D_0.priv$. D_n is a member of the role $A_n.r_n$. The inputs are a string tuple $[D_0.priv, P_{D_0}, A_1.r_1, P_{D_1}, \ldots, P_{D_{n-1}}, A_n.r_n, P_{D_n}]$ representing the delegation chain for the requester D_n, and a proof F that is computed by the requester D_n. In the string tuple, $D_0.priv$ is the delegated privilege, P_{D_i} for $i \in [0, n-1]$ is the public key for the delegator D_i, $A_i.r_i$ for $i \in [1, n]$ is the role that receives the delegation from D_{i-1}, and P_{D_n} is the public key of the requester. The tuple may also contain public keys of role administrators $A_1, \ldots, A_n$. The verifier checks whether F correctly authenticates the delegation chain. This includes authentication of each delegation extension $P_{D_{i-1}} \xrightarrow{D_0.priv} A_i.r_i$, and entity D_i's affiliate role membership $A_i \xrightarrow{A_i.r_i} D_i$, for all $i \in [1, n]$. The requester D_n also needs to prove the possession of the private key corresponding to the public key P_{D_n}. D_n is granted $D_0.priv$ if the verification is successful, and denied if otherwise.

Affiliated role credentials can be issued using INITIATE operation by the administrator of a role. EXTEND operation is used to issue delegated role credentials. The delegation chain of a privilege grows at each delegation extension.

4.2 Implementation of role-based cascaded delegation protocol

We present an implementation of role-based cascaded delegation using the Hierarchical Certificate-based Encryption (HCBE) [18] scheme. Each entity has a public/private key pair generated on his own. A member of an affiliated role has an affiliated role credential, which contains a signature signed by the administrator of the role. The delegation credential in this protocol consists of an aggregate signature and a string tuple. Our role-based cascaded delegation protocol has five operations, which make use of the algorithms in the HCBE scheme [18] defined in Appendix A.

SETUP: This operation is to setup system parameters, public/private keys, role credentials that would be used in the system.

- The root of the system calls the **Setup** algorithm in the HCBE scheme [18], and obtains a set of public parameters denoted as *params*. Among other parameters in *params*, there are two collision-resistant hash functions H and H', a special constant π, and bilinear map [9] function $\hat{e}$.
- As in HCBE [18] scheme, each entity (organization or individual) D chooses a secret s_D as his private key, and computes the product $s_D\pi$ as its public key P_D.
- An organization A with the private key s_A certifies the members who have $A.r$ as their affiliated role. For each entity D who has the affiliated role $A.r$ and the public key P_D, organization A computes a role signature R_D by running **Certification_of_CA**($s_A, P_D \| A.r$) of HCBE (see Appendix A), where $\|$ denotes string concatenation. The output signature represents the role credential $A \xrightarrow{A.r} D$, and is given to entity D for proving the affiliated role membership.

INITIATE: Resource owner D_0 delegates the privilege $D_0.priv$ to the members of an affiliated role $A_1.r_1$. The private key s_{D_0} corresponds to the public key P_{D_0} of entity D_0. Entity D_0 does the following.

- Set the string $info_1 = P_{D_0} \| D_0.priv \| A_1.r_1 \| P_{A_1}$, where P_{A_1} is the public key of the role administrator A_1. Run **Certification_of_CA**($s_{D_0}, info_1$) in HCBE, which outputs an extension signature X_1. Define a string tuple $chain_1$ as $[D_0.priv,\ P_{D_0},\ A_1.r_1,\ P_{A_1}]$. Set the partial delegation credential C_1 for the role $A_1.r_1$ as $(X_1, chain_1)$. Credential C_1 is put on a directory server.

EXTEND: An entity D_i, whose role is $A_i.r_i$, further delegates $D_0.priv$ to role $A_{i+1}.r_{i+1}$. D_i uses his private key s_{D_i}, his role signature R_{D_i}, and the delegation credential C_i for the role $A_i.r_i$ to compute a partial delegation credential C_{i+1}. Entity D_i does the following.

- Parse the credential C_i as $(S_{Agg}, chain_i)$, where S_{Agg} is the aggregate signature of credential C_i and $chain_i$ is the corresponding string tuple. Set the string $info_{i+1} = P_{D_0} \| D_0.priv \| A_{i+1}.r_{i+1} \| P_{A_{i+1}}$, where P_{D_0} is the public key of the resource owner and $P_{A_{i+1}}$ is the public key of the role administrator A_{i+1}. Run **Aggregate**($s_{D_i}, info_{i+1}, R_{D_i}, S_{Agg}$) in HCBE, which outputs an aggregate signature S'_{Agg}.
- The string tuple $chain_{i+1}$ of credential C_{i+1} is the string tuple $chain_i$ appended with public key P_{D_i}, the role $A_{i+1}.r_{i+1}$, and the public key $P_{A_{i+1}}$. Set credential $C_{i+1} = (S'_{Agg}, chain_{i+1})$. The delegation credential C_{i+1} for the role $A_{i+1}.r_{i+1}$ is put on a directory server.

PROVE: The requester D_n with the role signature R_n and delegation credential C_i wants to use the delegated role $D_0.priv$. D_n is given a randomized statement T by the verifier D_0. The statement T contains some random information to prevent a replay attack. D_n does the following.

- Parse the credential C_n as $(S_{Agg}, chain_n)$, where S_{Agg} is the aggregate signature of C_n and $chain_n$ is the string tuple. Run **Aggregate**($s_{D_n}, T, R_{D_n}, S_{Agg}$) in HCBE, which outputs an aggregate signature S'_{Agg}. Set the string tuple $chain'_n$ to $chain_n$ appended with the public key P_{D_n}. Set the proof F to be $(S'_{Agg}, chain'_n, T)$. Send the proof F to the verifier D_0.

VERIFY: The verifier D_0 verifies the proof F submitted by the requester D_n as follows.

- Parse F as $(S'_{Agg}, chain'_n, T)$, where S'_{Agg} is an aggregate signature, $chain'_n$ is a string tuple, and T is a statement. Parse the string tuple $chain'_n$ as $[D_0.priv,\ P_{D_0},\ A_1.r_1,\ P_{A_1},\ \ldots,\ A_n.r_n,\ P_{A_n},\ P_{D_n}]$, where for $i \in [0, n-1]$ P_{D_i} is the public key of delegator D_i, $A_{i+1}.r_{i+1}$ is the role receiving the delegation from D_i, $P_{A_{i+1}}$ is the public key of role administrator A_{i+1}, and P_{D_n} is the public key of the requester.

- Encrypt a message M as follows. Choose a random number r. Set the ciphertext $\mathit{Ciphertext} = [r\pi, V]$, where π is one of the public parameters, $V = M \oplus H'(g^r)$, and

$$g = \hat{e}(P_{D_n}, H(T))\Pi_{i=0}^{n-1}\hat{e}(P_{D_i}, H(P_{D_0}\|D_0.priv\|A_{i+1}.r_{i+1}\|P_{A_{i+1}}))\Pi_{i=1}^{n}\hat{e}(P_{A_i}, H(P_{D_i}\|A_i.r_i))$$

  The value g is the product of multiple bilinear map [9] functions whose inputs are the public key of a signer and the hash digest of the signed message. H' is another hash function in the system parameters $params$. $\oplus$ denotes bit-wise XOR operation. T is the message that D_n signs.

- Run **Decryption**($\mathit{Ciphertext}, S'_{Agg}$) in HCBE to decrypt ciphertext *Ciphertext* using S'_{Agg}. Compare the output M' of the decryption with the original message M. The request is granted if $M = M'$, denied if otherwise. The correctness of the verification is in Appendix B.

Delegation to intersection of roles [20], for example $A_1.r_1 \cap A_2.r_2$, may be realized by extending one delegation to a string that represents an intersection of roles, rather than one role. To extend or prove such a delegation, an entity needs to aggregate two, rather than one, role signatures into a delegation credential. Additional fields can be added by the delegator to a delegation credential to increase the expressiveness, one of them being the expiration date of a delegation.

5 Discussion

Revocation of delegations can be handled by having each resource owner maintaining a revocation server, using known techniques such as authenticated dictionary techniques [19, 24] and authenticated third-party publication techniques [22].

5.1 Privacy and security

In our role-based cascaded delegation model, only the credentials that are necessary for the verification of delegation chain are revealed. Unrelated credentials are not discovered or touched. This is a significant improvement in the privacy protection compared to other delegation models. Furthermore, our implementation provides strong protection of sensitive signatures, because individual signatures can be verified without being disclosed. This is not achievable in conventional signature schemes, such as RSA signature scheme. The security of HCBE guarantees that an attacker cannot forge a valid aggregate signature consisting n individual signatures, even if he possesses $n-1$ of the required private keys [10].

5.2 Scalability

The abstraction of roles in our model greatly reduces the potential large number of delegation credentials, and makes the model scalable. Because the partial delegation credentials issued by delegator cannot be directly used for accessing resources, they may be stored at public directory servers. Members of a role can query the directory to retrieve the partial credential. The implementation scales under large number of credential receivers. The delegation is decentralized. Individuals, who have qualified roles, can make delegations of the roles without the assistant of administrators. In collaboration environment where coalitions form dynamically, this feature gives great flexibility to the resource sharing.

An entity in the system is not required to store all possible delegation chains in the proof graph that connects the original issuer with him. For a given privilege, only one delegation credential is suffice. So the total number of delegation credentials in the role-based cascaded delegation is bounded by $\mathcal{O}(NM)$, where N is the total number of entities, and M is the total number of delegated privileges.

5.3 Efficiency comparisons

We compare our implementation with the implementation using the RSA signature scheme [2]. We consider 1024-bit modulus RSA scheme, in which the size of the public key is slightly more than 1024-bit and the size of a signature is 1024-bit.

5.3.1 Prove and extend

For the same level of security as 1024-bit modulus in RSA, the signature in our implementation is about 170-bit long, and so are all the public keys [12]. For a delegation chain of length n, i.e. having n delegations on the chain, the string tuple on a complete delegation credential in both implementations contains the delegated privilege, public keys of n delegators and n role administrators, n roles receiving the delegations, and the public key of the requester. Suppose the length of a role name is 100 bits, and the delegated privilege has the same size as a role name. The total size of the credential in our implementation is $170 + 170(2n + 1) + 100(n + 1) = 440n + 440$ bits. For RSA signature scheme, such a delegation credential contains $2n$ more signatures, and the total size is more than $1024(2n + 1) + 1024(2n + 1) + 100(n + 1) = 4196n + 2148$ bits.

For a delegation chain of length 20, the size of delegation credential in RSA is more than 86 Kbits, and in our implementation is 9.2 Kbits. Smart cards with the microprocessor typically have 32 KBytes (256 Kbits) EEPROM storage [14, 2]. Our saving in the credential size clearly shows the advantage in the number of credentials that can be stored by computational devices with small storages.

For 20 Kbits per second connection and a delegation chain of length 20, the time for transmitting the entire RSA credentials to the verifier in the PROVE operation takes $(4196 \times 20 + 2148)/20000 = 4.30$ seconds. The time in our implementation takes $(440 \times 20 + 440)/20000 = 0.46$ seconds. For small mobile devices with limited communication bandwidth, the saving in the credential size in our implementation allows the credentials to be transmitted faster.

The above analysis also applies to the EXTEND operation. In addition, generating a signature in our implementation requires only 3.57 ms to compute on a Pentium III 1 GHz, and is faster than generating a signature in RSA scheme, which requires 7.90 ms for 1007 bits private key on the same machine [5]. For EXTEND and PROVE operations, individual users just need to store the hash function H in the system parameters *params* in HCBE scheme [9].

5.3.2 Verify

The running time for verifying an aggregate signature associated with a delegation chain is linear in the number of single signatures aggregated, i.e. the length of the chain. Verification of one single signature in HCBE scheme is slow (about 50 ms on PIII 1 GHz), compared to RSA signature verification (0.40

[2]To prevent substitution attacks [29], this should be a nested RSA signature scheme, where a delegator signs on not only a delegation statement but also the signature obtained from his parent entity on the chain. We simplify it here, as the analysis is the same.

ms on the same machine) [5]. Nevertheless, in our implementation only the servers of resource owner, which are typically powerful, have to performs delegation chain verification. Therefore, the slowdown in the running time would be less noticeable, and does not affect the individual users who may have less powerful computational devices.

6 Related Work

In Table 1, several properties of our delegation model and existing delegation models that address delegation chain problems are compared.

Properties	Ours	*RT* framework [21]	KeyNote [7]	SPKI [1]	Hier. Token [15]
Cascaded	Yes	No	No	No	Yes
Storage	Distributed	Distributed	Centralized	Centralized	Distributed
Chain discovery	Not required	Required	Required	Required	Not required
Third-party	Not required	Required	N/A	N/A	Not required
Role-based	Yes	Yes	No	No	No
Cred. pass-down	Not required	N/A	N/A	N/A	Required
Size of cred.	$\mathcal{O}(l)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(l)$	$\mathcal{O}(l)$
Num of sig.	$\mathcal{O}(1)$	$\mathcal{O}(l)$	$\mathcal{O}(l)$	$\mathcal{O}(l)$	$\mathcal{O}(l)$
Privacy	Strong	Weak	Weak	Weak	Weak
Cryptographic op.	Pairing [5]	N/A	N/A	N/A	Exponentiation

Table 1: Comparisons of parameters in delegation systems that address the delegation chain issue. *Hier. Token* refers the hierarchical delegation protocol [15]. We denote with l is the number of entities on a delegation chain. *Third-party* means whether the delegation chain verification algorithms require third-party (intermediate entities) participation. *Cred. pass-down* refers to whether the delegation credential of the delegator has to be passed down to the delegatee. *Size of cred.* refers to the size of a delegation credential. *Num of sig.* means the number of signatures to be verified for a delegation credential chain. *Privacy* represents the degree of user privacy protection offered. *Cryptographic op.* is the cryptographic operation in the delegation and verification algorithms.

The *RT* framework is a family of Role-based Trust management languages for representing policies and credentials in decentralized authorization [20]. We have already compared our design with the credential chain discovery algorithms in *RT* framework at various places in the paper. The PolicyMaker [8] and KeyNote [7] trust management systems authorize decentralized access by checking the proof of compliance. SPKI/SDSI (Simple Public Key Infrastructure/Simple Distributed Security Infrastructure) is a public-key infrastructure emphasizing decentralized name space and flexible authorization [13, 1, 16, 4]. As noted earlier, KeyNote and SPKI/SDSI do not define explicit role abstractions, and assume that all of the certificates are available to the discovery algorithms.

There are several cascaded delegation [28] schemes for the proxy authentication and authorization, including nested signature schemes [29], delegation keys [25], and a combined approach [15]. These schemes do not consider delegations to roles, and the delegation credentials are not as compact as ours. The security framework for Java-based computing environment by Nagaratnam and Lea uses roles in chained delegations to simplify management of privileges, however, their delegations are made to individuals rather than to roles. Their term of cascaded delegation has different meanings from ours, and refers to delegations where all the privileges of each of the preceding entities on the chain are inherited by the delegatee.

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] R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed System.* Wiley, 2001.

[3] T. Aura. Comparison of graph-search algorithms for authorization verification in delegation networks. In *2nd Nordic Workshop on Secure Computer Systems NORDSEC'97*, November 1997.

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

[5] P. S. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *Proceedings of Crypto 2002*, volume 2442 of *Lectures in Computer Science*, pages 354–368. Springer-Verlag, 2002.

[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 Conference 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] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT 2003*, pages 416–432, 2003.

[11] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. A survey of two signature aggregation techniques. *CryptoBytes*, 6(2), 2003.

[12] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In *Proceedings of Asiacrypt 2001*, volume 2248 of *Lectures in Computer Science*, pages 514–532. Springer-Verlag, 2001.

[13] 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.

[14] DATAKEY. The model 330J smart card white paper. `http://www.datakey.com/resource/whitePapers/Model330JWhitePaperV1.pdf`.

[15] Y. Ding, P. Horster, and H. 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.

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

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

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

[19] 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.

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

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

[22] 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.

[23] N. Nagaratnam and D. Lea. Secure delegation for distributed object environments. In *Proceedings of the 4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS)*, April 1998.

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

[25] B. C. Neuman. Proxy-based authentication and accounting for distributed systems. In *International Conference on Distributed Computing Systems*, pages 283–291, 1993.

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

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

[28] K. R. Sollins. Cascaded authentication. In *Proceedings of 1988 IEEE Symposium on Research in Security and Privacy*, pages 156–163, 1988.

[29] V. Varadharajan, P. Allen, and S. Black. An analysis of the proxy problem in distributed systems. In *Proceedings of 1991 IEEE Symposium on Research in Security and Privacy*, pages 255–275, 1991.

[30] 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.

A Hierarchical Certificate-based Encryption

The Hierarchical Certificate-based Encryption (HCBE) scheme by Gentry [18] has six algorithms, **Setup**, **Certification_of_CA**, **Certification_of_Bob**, **Aggregation**, **Encryption**, and **Decryption**.

Setup(): A set of system parameters *params* is generated, and will be used in all the operations of the scheme. Among other parameters, *params* contain two cryptographic hash functions H and H', a bilinear map $\hat{e}$, and a constant π with certain properties. A bilinear map [9] is a mapping function $\hat{e}(x, y)$ that takes two inputs x and y, and outputs a value.

An entity D chooses his private key s_D, which is a positive integer. Entity D computes and publishes his public key $s_D\pi$ by multiplying s_D with the parameter π. In what follows, public key is expressed in the form of a product. The key pair may be used for signing and secure email purposes. In the following, we suppose Bob is at level n, that his public key is $s_n\pi$ and private key is s_n, and that the CAs above him have public keys $s_i\pi$ and private keys s_i for $1 \leq i \leq n-1$.

Certification_of_CA($s_i\pi, \mathit{info}_{i+1}$): CA at i-th level certifies the public key of the CA at at level $i+1$ by computing a signature. The first input is the public key of CA$_i$, and the second input is a string info_{i+1} that contains the public key $s_{i+1}\pi$ of CA$_{i+1}$. The string info_{i+1} may also contain information such as expiration date, etc. CA$_i$ first computes a hash digest $H(s_i\pi \| \mathit{info}_{i+1})$, and then multiplies the digest with his private key s_i. Recall H is one of the hash function in the system parameters *params*. The output signature $s_i H(s_i\pi \| \mathit{info}_{i+1})$ is given to CA$_{i+1}$.

Certification_of_Bob($s_{n-1}\pi, \mathit{info}_n$): Bob asks his parent CA, CA$_{n-1}$, to certify his public key by computing a signature. The first input is the public key of CA$_{n-1}$, and the second input is a string info_n that contains the public key $s_n\pi$ of Bob. CA$_{n-1}$ computes the signature as $s_{n-1}H(s_{n-1}\pi \| \mathit{info}_n)$, which is the multiplication of CA$_{n-1}$'s private key s_{n-1} with the hash digest of the signer's public key and info_n. The output signature is given to Bob.

Aggregation($s_n, \mathit{info}_n, \mathit{sig}_2, \ldots, \mathit{sig}_n$): Bob uses his private key s_n and the public key certificates on his chain to compute an aggregate signature, which will be used as his decryption key. The inputs to this algorithm are Bob's private key [3] s_n, the string info_n that is used for certifying Bob's public key, and a number of signatures [4] that contains the public key certificate signatures associated with his chain. Recall that the public key certificate signature sig_i for the entity at level i is of the form $s_{i-1}H(s_{i-1}\pi \| \mathit{info}_i)$, for $2 \leq i \leq n$. Bob first computes a signature on the string info_n, $s_n H(\mathit{info}_n)$. He then aggregates this signature with all the input signatures simply by adding them together, $S_{Agg} = s_n H(\mathit{info}_n) + \sum_{i=2}^{n} \mathit{sig}_i$. The output S_{Agg} is Bob's decryption key.

Encryption($M, \mathit{info}_1, \ldots, \mathit{info}_n$): Alice computes an encrypted message to send to Bob. The inputs are a message M, string info_i of the entity at level i on Bob's chain for $1 \leq i \leq n$. Recall that info_i contains the public key $s_i\pi$ of CA$_i$ for $1 \leq i \leq n-1$, and info_n contains the public key $s_n\pi$ of Bob. Alice encrypts a message M using the public keys and a random number r. The ciphertext C consists of two values, $C = [r\pi, V]$. The first component is the product of the random number r and public parameter π. The second component is computed as $V = M \oplus H'(g^r)$, where $g = \hat{e}(s_n\pi, H(\mathit{info}_n))\Pi_{i=1}^{n-1}\hat{e}(s_i\pi, H(s_i\pi \| \mathit{info}_{i+1}))$, $\oplus$ denotes bit-wise XOR operation, H' is the other cryptographic hash function in the system parameters

[3]This algorithm is run by Bob, so the private key is safe.
[4]Aggregate algorithm can take any number of signatures.

params, and $\hat{e}$ is the bilinear map in *params*. g is the product of n bilinear map computations, whose inputs are a public key and a hash digest. The output ciphertext C is sent to Bob.

Decryption(C, S_{Agg}): Bob decrypts the ciphertext C to retrieve the message using his aggregate signature S_{Agg}. Bob first parses the ciphertext C as two values (U, V). He then computes the message $M = V \oplus H'(\hat{e}(U, S_{Agg}))$. The bilinear map $\hat{e}$ takes two inputs: one is the first component U of the ciphertext C, and the other is Bob's aggregate signature S_{Agg}. The output is a message M.

The correctness of the decryption is shown in the Appendix A.1.

A.1 Correctness of HCBE

Bob's aggregate signature is $S_{Agg} = s_n H(\mathit{info}_n) + \sum_{i=1}^{n-1} s_i H(s_i\pi, \mathit{info}_{i+1})$. The bilinear map computation in the **Decryption** is as follows.

$$\begin{aligned}
\hat{e}(U, S_{Agg}) &= \hat{e}(r\pi, s_n H(\mathit{info}_n) + \sum_{i=1}^{n-1} s_i H(s_i\pi, \mathit{info}_{i+1})) \\
&= \hat{e}(r\pi, s_n H(\mathit{info}_n))\Pi_{i=1}^{n-1}\hat{e}(r\pi, s_i H(s_i\pi, \mathit{info}_{i+1})) \\
&= \hat{e}(s_n\pi, H(\mathit{info}_n))^r \Pi_{i=1}^{n-1}\hat{e}(s_i\pi, H(s_i\pi, \mathit{info}_{i+1}))^r \\
&= g^r
\end{aligned}$$

where $g = \hat{e}(s_n\pi, H(\mathit{info}_n))\Pi_{i=1}^{n-1}\hat{e}(s_i\pi, H(s_i\pi, \mathit{info}_{i+1}))$. Therefore, $V \oplus H'(\hat{e}(U, S_{Agg})) = V \oplus H'(g^r) = M$.

B Correctness of our implementation

The bilinear map computation in the **Decryption** is as follows.

$$\begin{aligned}
\hat{e}(U, S_{Agg}) &= \hat{e}(r\pi, s_{D_n} H(T) + \sum_{i=0}^{n-1} s_{D_i} H(P_{D_0} \| D_0.\mathit{priv} \| A_{i+1}.r_{i+1} \| P_{A_{i+1}}) + \sum_{i=1}^{n} s_{A_i} H(P_{D_i} \| A_i.r_i)) \\
&= \hat{e}(r\pi, s_{D_n} H(T))\Pi_{i=0}^{n-1}\hat{e}(r\pi, s_{D_i} H(P_{D_0} \| D_0.\mathit{priv} \| A_{i+1}.r_{i+1} \| P_{A_{i+1}}))\Pi_{i=1}^{n}\hat{e}(r\pi, s_{A_i} H(P_{D_i} \| A_i.r_i)) \\
&= \hat{e}(s_{D_n}\pi, H(T))^r \Pi_{i=0}^{n-1}\hat{e}(s_{D_i}\pi, H(P_{D_0} \| D_0.\mathit{priv} \| A_{i+1}.r_{i+1} \| P_{A_{i+1}}))^r \Pi_{i=1}^{n}\hat{e}(s_{A_i}\pi, H(P_{D_i} \| A_i.r_i))^r \\
&= g^r
\end{aligned}$$

where $g = \hat{e}(P_{D_n}, H(T))\Pi_{i=0}^{n-1}\hat{e}(P_{D_i}, H(P_{D_0} \| D_0.\mathit{priv} \| A_{i+1}.r_{i+1} \| P_{A_{i+1}}))\Pi_{i=1}^{n}\hat{e}(P_{A_i}, H(P_{D_i} \| A_i.r_i))$. Therefore, $V \oplus H'(\hat{e}(U, S_{Agg})) = V \oplus H'(g^r) = M$.