Role-Based Cascaded Delegation*

Roberto Tamassia
Computer Science Department
Brown University
Providence, RI 02912
rt@cs.brown.edu

Danfeng Yao
Computer Science Department
Brown University
Providence, RI 02912
dyao@cs.brown.edu

William H. Winsborough
Center for Secure Information Systems
George Mason University
Fairfax, VA 22030-4444
wwinsbor@gmu.edu

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, 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.

Categories and Subject Descriptors

D.4.6 [**Operating System**]: Security and Protection—*Access Controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*

General Terms

Security

Keywords

Access Control, RBAC, Delegation, Trust Management

1. INTRODUCTION

Trust management (TM) is an approach to access control in environments where entities that are not in the same security domain need to share resources. Several TM systems have been proposed in recent years, e.g., PolicyMaker [4], KeyNote [3], SPKI/SDSI [9], and the *RT* framework [20].

The notion of delegation is essential in transferring trust and authorization in TM systems. Delegation chains trace sequences of entities, starting from the resource owner and including entities authorized by (though possibly unknown to) the owner. These entities play a central part in decentralized authorization by providing the credentials that represent their own delegation acts, which enable the delegation chain to be verified. Collecting and verifying these credentials incurs communication and computation costs, as does checking that together the credentials provide proof that a given user or software agent is authorized. In this paper, we present techniques that can be used to significantly reduce these costs, which we expand upon in the next two sections. This introduction then illustrates our model of delegation, outlines our approach to providing an efficient implementation of that model, states the current contributions, and presents the organization of the remainder of the paper.

*Work supported in part by NSF grants IIS–0324846 and CCR–0325951, and by a research gift from Sun Microsystems, Inc.

SACMAT'04, June 2–4, 2004, Yorktown Heights, New York, USA.

1.1 Credential accumulation

Most prior work that addresses the problem of determining whether credentials prove an entity's resource request is authorized [3, 4, 9] assumes that all potentially relevant credentials are available in one central storage. There are some exceptions. QCM [18] and SD3 [19] are two trust-management systems that consider distributed storage of credentials. A limitation of the approach in QCM and SD3 is assuming that issuers initially store all the credentials, which may be impractical for some applications. This limitation was addressed by Li *et al.* [21], who presented goal-directed credential chain discovery algorithms that support a more flexible distributed storage scheme in which credentials may be stored by their issuer, their recipient (also called their "subject"), or both. The algorithms dynamically search for relevant credentials from remote servers to build a proof of authorization.

While storing credentials with their issuers or recipients supports flexible delegation models such as those captured in the *RT* framework, in many cases such flexibility is unnecessarily costly. Moreover, distributed credential discovery algorithms also have the following two potentially problematic characteristics. First, they require credential issuers or their responders (credential servers) to be available and participate in the computation. Second, those issuers must make available a potentially large number of their own credentials, which may contain sensitive information that should not be made public, and many of which are likely to be irrelevant to any given authorization decision. For these reasons, storing credentials with their issuers or recipients may often be inappropriate.

In many contexts, it is more practical to require creden-

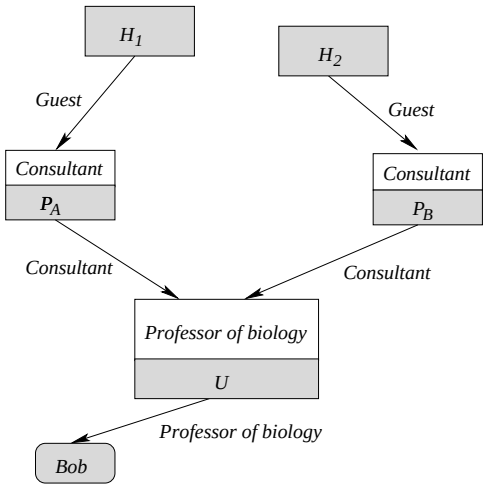

Figure 1: A schematic drawing of the delegation relationships described in Section 1.1. Arrows represent the direction of the delegation action.

tials to be accumulated by the entities whose authorization they prove. Those entities can then present the accumulated credentials to service providers so as to prove their own authorization. They can also provide accumulated credentials, along with delegation credentials they themselves issue, to recipients of such delegations. This *accumulation* approach to credential collection offers significant reduction in communication costs compared with traditional credential collection techniques.

Consider the following example that is illustrated in Figure 1. Bob occupies the role *professor of biology* at university U, and has a *role credential* issued by U that documents this. The role *professor of biology* at U is delegated the role *consultant* by two pharmaceutical companies P_A and P_B, respectively, which means biology professors are eligible to act as consultants. To achieve this, each company 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 credential servers, respectively.

In order for Bob to use his delegated role *guest* of H_1, credentials that prove his membership in that role must be collected. If credential storage is distributed, this can be costly. For example, in Li's 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. Thus, the number of messages exchanged to discover a single delegation path is proportional to the number of edges of the entire graph. The potentially high communication cost incurred in discovering delegation chains is also reported by Aura [1]. We will apply the accumulation approach for credential collection to this example in Section 1.3 when we introduce our role-based cascaded delegation model.

The definition of delegation chain in our model is slightly different from the delegation chain in some role-based delegation systems [9, 21]. Namely, in our model, a delegation chain represents the path on which a delegated privilege (role or permission) is transferred among roles and entities.

1.2 Delegation chain verification

The verification of a *role-based* delegation chain such as the one pictured in Figure 1 may be quite expensive if the chain is represented in a typical trust management-system implementation. 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 for ensuring 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. In the accumulation approach, an intermediate delegator passes 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 not want to disclose it to the delegatee. This sensitive signature issue is solved in our implementation of role-based cascaded delegation using Hierarchical Certificate-based Encryption [15] scheme in Section 5.

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 [24, 29]. The distributed cascaded delegation problem is essentially to design a delegation mechanism that efficiently verifies a hierarchical delegation chain. In the cascaded delegation model, a delegation recipient E may further extend the delegated privilege to another entity E', and the delegation credentials of E are passed to entity E' along with the delegation certificate signed by E as the issuer. 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 [14, 28].

Our role-based cascaded delegation assumes that credentials are collected by authorized agents through the accumulation scheme discussed in Section 1.1, which eliminates the need for distributed delegation chain discovery. Delegations can be issued to roles of different administrative 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 C that explicitly assigns the privilege to role r. Members of the role r are allowed to further delegate the privilege to another role r' as follows. A member D of the role r uses the delegation credential C to generate a delegation credential C'. C' comprises multiple component credentials, which include the credential of the current delegation transaction, the credential C from the preceding delegation, and the role membership credential of the delegator D. The verifier can make the authorization decision based on delegation credential C' and the role membership credential of the requester.

For example, hospital H delegates its role *guest* to the role *consultant* at company P in credential C. This delegation means that a member of *consultant* at company P is also a member of *guest* at hospital H and can access resources of H

that are associated with the role *guest*. John is a member of the role *consultant* at company P and has the corresponding role credential R. John may further delegate the role *guest* at H to the role *professor* at university U in credential C'. This delegation means that a member of *professor* at university U is also a member of *guest* at hospital H. Credentials C, R, and C' constitute the delegation credential for the role *professor* at U. The role credential R proves that John is indeed a member of *consultant* at company P and therefore is entitled to issue delegations.

A schematic drawing of the components in the delegation credential issued to *professor* at U in the role-based cascaded delegation model is shown in Figure 2.

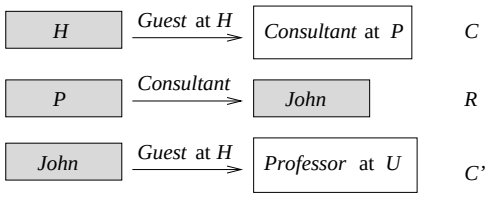

Figure 2: A schematic drawing of the components of the delegation credential for the role *professor* at university U in the role-based cascaded delegation model.

Our role-based cascaded delegation model can be used to facilitate large-scale dynamic sharing of resources in decentralized, pervasive collaborative environments. 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 motivating the design of our role-based cascaded delegation is the size and number of credentials an entity has to store or transmit for delegation and verification purposes. In existing cascaded delegation protocols, delegation credentials are lengthy because verification of a delegation chain requires checking a number of signatures linear in the length of the chain, where the length is defined as the number of delegations on the chain. Conventional signature schemes, such as RSA [27] and DSA [12], produce relatively long signatures compared to the security they provide. 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 about $2n$, because in addition to verifying each of the delegation transactions, one must verify arbitrarily large proofs that the intermediate delegators are members of the required roles. 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 is not known in conventional cryptosystems, such as RSA [7, 23]. 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 entities who have limited computational power and communication bandwidth, the implementation of role-based cascaded delegation using conventional credentials is inefficient. Another potential issue in implementing role-based cascaded delegation is privacy. 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) [15] scheme. Using HCBE allows the authentication information of the entire role-based delegation chain to be 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 underlying short signature [8] and aggregate signature [6] techniques. Sensitive signatures are also protected in this implementation, because an individual signature, once aggregated, can be verified without being disclosed.

1.5 Contributions

In this paper, we formalize the role-based cascaded delegation model, which combines role-based trust management with a cascaded delegation mechanism. This model supports flexible and scalable decentralized role-based delegations, which may be issued without the participation of the administrator of an organization. Cascaded delegation uses the accumulation approach to credential collection, which reduces communication and computation costs in comparison to collection based on retrieval from distributed storage. User privacy is protected in this model, because no unnecessary delegation credentials are revealed.

We also present an efficient implementation of role-based cascaded delegation using the Hierarchical Certificate-based Encryption (HCBE) scheme [15]. In this implementation, the authentication information for an entire role-based delegation chain is captured by one short signature of constant size. This implementation also offers further privacy protection because individual signatures can be verified without being disclosed.

1.6 Organization of the paper

The rest of this paper is organized as follows. Our model for roles and their delegation scope is given in Section 2. Preliminary definitions and our language model are provided in Section 3. In Section 4, an example of cascaded delegation is presented. The role-based cascaded delegation protocol and its implementation based on HCBE are described in Section 5. In Section 6, we address the issues of revocation, privacy, security, scalability, and efficiency for our model and implementation. A comparison of role-based cascaded delegation with existing decentralized trust management approaches is given in Section 7. Section 8 contains the conclusions and the future work. An overview of the HCBE scheme is provided 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 the role, that role is said to be an *affiliated role* of D. (This type of role is usually obtained through the affiliation with an organization, and thus the name.) If a role credential of D is instead issued through delegation and

signed by entities other than the administrator of the role, that role is called a *delegated role* of D.

The following example illustrates the difference between affiliated and delegated roles. Bob has a credential signed by university U for the role *professor* at U, denoted $U.professor$. Thus, role $U.professor$ is an affiliated role of Bob. Alice is delegated the role $U.professor$ by Bob. However, Alice does not have a credential signed by U for $U.professor$. Thus, $U.professor$ is a delegated role of Alice.

An affiliated role and a delegated role have different access scopes. Delegations to a role r of an organization only apply to those entities who have r as an affiliated role. In the above example, if a privilege is delegated to role $U.professor$, then Bob is entitled to this privilege, whereas Alice is not. The delegated role $U.professor$ of Alice only allows her to access resources controlled by U.

Our delegation model for roles is different from conventional delegation models, where delegations to a role *automatically* propagate to *all* the entities that are delegated the role. For example, if hospital H delegates the right of reading a patient's medical record to the role $U.professor$, Alice would be entitled to this privilege in conventional delegation models, but not in our model. For sensitive data such as medical records, the automatic propagation of delegations to unknown roles may not always be desired by the resource owner. In comparison, our delegation model allows easy management of delegations for resource owners.

To support flexible decentralized delegation, we give to both role types (affiliated and delegated) the capability of delegating the role to other roles. Thus, in the above example, both Bob and Alice are able to delegate role $U.professor$ to other roles.

3. PRELIMINARIES

In this section, we define our terminology and language model, and give a brief overview of the HCBE scheme [15].

3.1 Terminology

As in the *RT* framework [21], we define an *entity* to be either an organization or an individual. An entity may issue credentials and make requests. Also, an entity may have one or more affiliated roles or delegated roles, which are authenticated by role credentials. An *affiliated role credential* is the credential for an affiliated role, and is signed by the administrator of the role. Similarly, a *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 credential* is generated and signed by a delegator on delegation transaction information, such as identities of the delegator and delegatee, and the delegated privilege. An *extension signature* is the signature on an extension credential. 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 her private key. A *complete delegation credential* includes the identity signature of the requester, extension signatures, and role signatures. A *partial delegation credential* is a delegation credential issued to a role. It cannot be used by an individual for proving authorization, as it lacks the identity and role signatures of the requester.

3.2 Language model

A role r administered by entity A is denoted as $A.r$. Entity A is the administrator of role $A.r$. A role defines a group of entities who are members of this role. If an entity D has an affiliated role $A.r$, her *role credential* is denoted by $A \xrightarrow{A.r} D$, which indicates that D is assigned role $A.r$ by the role administrator A. Entity D can delegate role $A.r$ to a role $B.s$ (administered by B) by issuing an *extension credential*, which is denoted by $D \xrightarrow{A.r} B.s$. In turn, any member entity E of role $B.s$ can further delegate role $A.r$ to a role $C.t$ (administered by C). The corresponding extension credential is denoted by $E \xrightarrow{A.r} C.t$.

3.3 HCBE

The *Hierarchical Certificate-based Encryption* (HCBE) scheme [15] is 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. The CA hierarchy consists of a root CA and lower-level CAs. Higher-level 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.

HCBE is based on the aggregate signature scheme [6, 8], which supports aggregation of multiple signatures on distinct messages from distinct users into one short signature. The HCBE scheme [15] 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 certifying the public keys of CAs, and the other for an individual. A description of these algorithms is given in Appendix A.

4. EXAMPLE SCENARIO

In this section, we describe a delegation example for the role-based cascaded delegation model. Suppose a collaboration project is established between a hospital H and a medical school M. To facilitate the collaboration, the hospital initiates a delegation chain and delegates its role $H.guest$ to the affiliated role $M.professor$ at the medical school. Hospital H is the administrator of the role $H.guest$. The delegation is expressed in the partial delegation credential (1), using the notation described in Section 3.2.

$$H \xrightarrow{H.guest} M.professor \quad (1)$$

In credential (1), hospital H is the original issuer, $H.guest$ is the delegated privilege, and $M.professor$ is the role that receives the delegation.

The hospital H allows members of the role $M.professor$ to further delegate $H.guest$ role to whomever they deem necessary to accomplish the project. Bob is a professor at M and has an affiliated role credential (2).

$$M \xrightarrow{M.professor} \text{Bob} \quad (2)$$

For a task in the collaboration project, Bob works with a lab L. Lab L is independent of the school M and is unknown to the hospital H. Lab L defines a research assistant role $L.assistant$. In order for members of the role $L.assistant$

to work on the task and utilize the resources of the hospital H, Bob delegates the role $H.guest$ to the affiliated role $L.assistant$. In the cascaded delegation model, Bob issues a partial delegation credential (3) by extending the delegation credential (1) to role $L.assistant$.

$$(H \xrightarrow{H.guest} M.professor), (M \xrightarrow{M.professor} \text{Bob}),$$
$$(\text{Bob} \xrightarrow{H.guest} L.assistant) \quad (3)$$

Credential (3) also includes Bob's role credential (2) for proving that he is allowed to delegate $H.guest$. (3) is a partial delegation credential for role $L.assistant$.

Alice is a research assistant in lab L, and has an affiliated role credential (4) issued by lab L to prove this role membership.

$$L \xrightarrow{L.assistant} \text{Alice} \quad (4)$$

To prove that she has the hospital's delegated *guest* role, Alice obtains the delegation credential (3) for role $L.assistant$ from a credential server, and aggregates it with her affiliated role credential (4). This gives credential (5).

$$(H \xrightarrow{H.guest} M.professor), \quad (M \xrightarrow{M.professor} \text{Bob}),$$
$$(\text{Bob} \xrightarrow{H.guest} L.assistant), \quad (L \xrightarrow{L.assistant} \text{Alice}) \quad (5)$$

Credential (5) and the identity signature of Alice yield a complete delegation credential for Alice. The hospital H makes the authorization decision by verifying each of the components of credential (5) and Alice's identity signature. Our implementation using HCBE scheme described in Section 5.2 allows the credentials to have just one signature, as opposed to the linear number of signatures in the length of the chain.

5. ROLE-BASED CASCADED DELEGATION

In this section, we first describe the role-based cascaded delegation protocol and then show an efficient implementation of this protocol using the HCBE scheme [15]. In what follows, a role r represents an affiliated role.

5.1 Protocol

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

- INITIATE(P_{D_0}, s_{D_0}, $D_0.priv$, $A_1.r_1$, P_{A_1}): This operation is run by the administrator D_0 of a privilege $D_0.priv$ to delegate $D_0.priv$ to an affiliated role $A_1.r_1$. This operation initiates a delegation chain for privilege $D_0.priv$. Inputs are the public key P_{D_0} of entity D_0, the corresponding private key s_{D_0}, the delegated privilege $D_0.priv$, the role name $A_1.r_1$, and the public key P_{A_1} of role administrator A_1. The output is a partial delegation credential C_1 for the role $A_1.r_1$, represented as
$$D_0 \xrightarrow{D_0.priv} A_1.r_1.$$
The statement of C_1 includes the public key P_{D_0}, the privilege $D_0.priv$, and information about the role $A_1.r_1$ such as the role name and the public key of the administrator A_1. The certificate is signed using the private key s_{D_0}. If the last argument is the public key of an individual, this operation can also be used for generating role certificates.

- EXTEND ($s_{D_n}, D_0.priv, C_n, R_{D_n}, A_{n+1}.r_{n+1}, P_{A_{n+1}}$):
This operation is run by an intermediate delegator D_n, who is a member of an affiliated role $A_n.r_n$, to extend the delegation of privilege $D_0.priv$ to the role $A_{n+1}.r_{n+1}$. The inputs are the private key s_{D_n} of the delegator D_n, the delegated privilege $D_0.priv$, the partial delegation credential C_n that delegates the privilege $D_0.priv$ to the role $A_n.r_n$, the role credential R_{D_n} of the delegator D_n, the role name $A_{n+1}.r_{n+1}$, and the public key $P_{A_{n+1}}$ of role administrator A_{n+1}. Credential C_n is retrieved from a credential server. The partial delegation credential C_n is a function of preceding extension and role credentials, which are denoted as:
$$(D_0 \xrightarrow{D_0.priv} A_1.r_1),$$
$$(A_1 \xrightarrow{A_1.r_1} D_1), \quad (D_1 \xrightarrow{D_0.priv} A_2.r_2),$$
$$\dots$$
$$(A_{n-1} \xrightarrow{A_{n-1}.r_{n-1}} D_{n-1}), \quad (D_{n-1} \xrightarrow{D_0.priv} A_n.r_n)$$
where D_0 represents the resource owner, 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 $i \in [1, n]$.

An extension credential $D_n \xrightarrow{D_0.priv} A_{n+1}.r_{n+1}$ is generated as an intermediate product of the operation EXTEND. Its statement contains information about the delegated privilege $D_0.priv$ and the role $A_{n+1}.r_{n+1}$. It is signed with the private key s_{D_n}. The final output of this operation is a partial delegation credential C_{n+1}, which is a function of the credential C_n, the role credential R_{D_n} denoted by $A_n \xrightarrow{A_n.r_n} D_n$, and the extension credential described above.

Credential C_{n+1} may simply be delegation credential C_n together with 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 [11, 26], and also passes down his role credential to members of the role $A_{n+1}.r_{n+1}$. In comparison, our implementation using HCBE [15] scheme provides a more efficient approach.

- PROVE($s_{D_n}, D_0.priv, R_{D_n}, C_n$):
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 requester D_n uses the partial delegation credential C_n and D_n's affiliated role credential R_{D_n}, denoted by $A_n \xrightarrow{A_n.r_n} D_n$, to prove that he is authorized the privilege $D_0.priv$. The inputs are the private key s_{D_n} of the requester D_n, the privilege $D_0.priv$, the affiliated role credential R_{D_n} of the requester, and the delegation credential C_n. Credential C_n is retrieved by the requester from a credential server. The operation produces a proof F, which contains delegation statements and corresponding signatures for verification. The private key s_{D_n} is for proving the authenticity of the public key P_{D_n} that appears on the role credential R_{D_n} of the requester.

- VERIFY(F):
This operation is performed by the resource owner 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 input is a proof F that is computed by the requester D_n. F contains signatures and a string tuple $[D_0.priv, P_{D_0}, A_1.r_1, P_{A_1}, P_{D_1}, \ldots, P_{D_{n-1}}, A_n.r_n, P_{A_n}, P_{D_n}]$ that consists of the components of a delegation chain for requester D_n. In the string tuple, $D_0.priv$ is the delegated privilege, for $i \in [1,n]$ $P_{D_{i-1}}$ is the public key for the delegator D_{i-1} whose affiliated role is $A_{i-1}.r_{i-1}$, $A_i.r_i$ is the role that receives the delegation from D_{i-1}, P_{A_i} is the public key of role administrator A_i, and P_{D_n} is the public key of the requester. The verifier checks whether the signatures in F correctly authenticates the delegation chain. This includes authentication of each delegation extension $D_{i-1} \xrightarrow{D_0.priv} A_i.r_i$, and entity D_i's affiliated role membership $A_i \xrightarrow{A_i.r_i} D_i$, for all $i \in [1,n]$. F also contains the proof of possession of private key s_{D_n} that corresponds to 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. The verifier may perform revocation checking at the VERIFY operation. Delegation revocation is discussed in Section 6.

5.2 Implementation

Role-based cascaded delegation can be implemented in a straightforward manner using the RSA signature scheme [27]. At each delegation, the delegator D computes an RSA signature on the delegation statement, and issues it to delegatees along with D's role signature (also an RSA signature). The delegation chain verification consists of verifying each of the above signatures.

We present a more efficient implementation of role-based cascaded delegation using the Hierarchical Certificate-based Encryption (HCBE) [15] scheme. In HCBE, 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 implementation role-based cascaded delegation protocol has five operations, which make use of the algorithms in the HCBE scheme [15] defined in Appendix A.

SETUP: This operation outputs the system parameters, public/private keys, and role credentials that will be used in the system.

- The root of the system calls the **Setup** algorithm of HCBE 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 a bilinear map $\hat{e}$ [5].
- 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 entities who have $A.r$ as an 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, where $\|$ denotes string concatenation. The output signature, representing the role assignment $A \xrightarrow{A.r} D$, is given to entity D for proving the affiliated role membership.

INITIATE: Resource owner D_0 delegates the privilege $D_0.priv$ to 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 credential 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 of 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. Signature S_{Agg} is a function of preceding extension and role signatures on the delegation chain. String tuple $chain_i$ contains the components of the delegation chain. 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 **Aggregation**$(s_{D_i}, info_{i+1}, R_{D_i}, S_{Agg})$ in HCBE, which outputs an aggregate signature S'_{Agg}.
- Define the string tuple $chain_{i+1}$ of credential C_{i+1} as the string tuple $chain_i$ appended with public key P_{D_i}, the role name $A_{i+1}.r_{i+1}$, and the public key $P_{A_{i+1}}$. Set credential C_{i+1} as $(S'_{Agg}, chain_{i+1})$. The partial delegation credential C_{i+1} for the role $A_{i+1}.r_{i+1}$ is put on a credential server.

PROVE: The requester D_n with the role signature R_{D_n} and delegation credential C_n wants to use the delegated privilege $D_0.priv$. D_n is given a random message T by the verifier D_0. The message 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 **Aggregation**$(s_{D_n}, T, R_{D_n}, S_{Agg})$ in HCBE, where s_{D_n} is the private key of D_n. This algorithm outputs an aggregate signature S'_{Agg}. Set the string tuple $chain'_n$ to be $chain_n$ appended with the public key P_{D_n} of D_n. Set the proof F to be $(S'_{Agg}, chain'_n, T)$, which is sent 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 message. 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 whose affiliated role is $A_i.r_i$, $A_{i+1}.r_{i+1}$ 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 $Ciphertext = [r\pi, V]$, where π is one of the public parameters, $V = M \oplus H'(g^r)$, where $g = g_1 g_2 g_3$ is a product of the following: $g_1 = \hat{e}(P_{D_n}, H(T))$, $g_2 = \Pi_{i=1}^{n} \hat{e}(P_{A_i}, H(P_{D_i} \| A_i.r_i))$, $g_3 = \Pi_{i=0}^{n-1} \hat{e}(P_{D_i}, H(P_{D_0} \| D_0.priv \| A_{i+1}.r_{i+1} \| P_{A_{i+1}}))$.
  The value g is the product of multiple bilinear map functions [5] whose inputs are the public key of a signer and the hash digest of the signed message. H and H' are the two hash functions in the system parameters $params$. $\oplus$ denotes bit-wise XOR operation. T is the message that D_n signs in PROVE.

- Run **Decryption**$(Ciphertext, S'_{Agg})$ 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.

A 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.

6. DISCUSSION

In this section, we discuss the main features of role-based cascaded delegation.

6.1 Privacy and security

In the 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 terms of privacy protection, over other delegation models that require extensive credential chain discovery as discussed in Section 1.

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 [7]. Also, the security of HCBE guarantees that an attacker cannot forge a valid aggregate signature consisting of n individual signatures, even if he possesses $n-1$ of the required private keys [6].

6.2 Scalability

The abstraction of roles in role-based cascaded delegation greatly reduces the potential for a large number of delegation credentials, and makes the model scalable. Because the partial delegation credentials issued by the delegators cannot be directly used for accessing resources, they may be stored at credential servers so that members of a role can query the server to retrieve the partial credential. Thus, our implementation scales up to a large number of credential receivers. Also, the delegation is decentralized. Individuals, who have qualified roles, can make delegations of the roles without the assistance of administrators. In collaboration environments where coalitions are formed dynamically, this feature greatly facilitates resource sharing.

An entity in the system is not required to store all possible delegation chains in the proof graph that connect the original issuer with him. Indeed, for a given privilege, only one delegation credential is sufficient.

6.3 Efficiency

We compare our HCBE-based implementation with the implementation using the RSA signature scheme [27] described at the beginning of Section 5.2. We consider a 1024-bit modulus RSA scheme, in which the size of the public key is slightly larger than 1024 bits and the size of a signature is 1024 bits long.

For the same level of security as 1024-bit modulus RSA, the signature and public key in our implementation can be as short as 170-bit long [8]. Observe that at each delegation extension of RBCD, the following information needs to be added to the delegation credential: the public key of the delegator, the role name of recipients, the public key of the role administrator, the signature on the role credential of the delegator, and the extension signature generated by the issuer. The analysis also applies to the AGGREGATE operation performed by the requester. Therefore, to authenticate a delegation chain of length n (i.e. having n delegations), the information required by the verifier includes the delegated privilege, the public keys of n delegators and n role administrators, n role names, the public key of the requester, along with $2n+1$ digital signatures.

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 the RSA signature scheme, such a delegation credential contains $2n$ additional signatures, and the total size is at least $1024(2n+1) + 1024(2n+1) + 100(n+1) = 4196n + 2148$ bits.

For example, consider a delegation chain of length 20. The size of the delegation credential in RSA is more than 86 Kbits, while in our implementation it is about 9.2 Kbits. Smart cards with a microprocessor typically have 32 KBytes (256 Kbits) EEPROM storage. Thus, our approach has a clear advantage in terms of the number of credentials that can be stored by smart cards and similar devices.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.

For a 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.

In addition, generating a signature in our implementation requires only 3.57 ms to compute on a 1 GHz Pentium III, and is faster than generating a signature in the RSA scheme,

which requires 7.90 ms for a 1007-bit private key on the same machine [2].

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. The verification of a signature in the HCBE scheme is slow (about 50 ms on a 1 GHz Pentium III) compared to RSA signature verification (0.40 ms on the same machine for a 1007 bits private key) [2]. Nevertheless, in our implementation only the servers of resource owners, which are typically powerful, have to performs delegation chain verifications.

6.4 Delegation renewal and revocation

At each delegation extension, the issuer can set an expiration date for the delegation, which may be earlier than the expiration dates of preceding delegations on the chain. For a delegation credential to be considered valid, none of the expiration dates has passed. Intermediate delegators may issue delegations with a short validity period, and then periodically renew them. Delegation renewal can be done in a hierarchical fashion as follows. To renew a delegation, a delegator E puts the renewed partial delegation credential on credential servers. Intermediate delegators that succeed to E may retrieve the renewed credential and update the corresponding delegations that are issued by them.

Delegation revocation before expiration can be handled by maintaining a revocation service, which can be efficiently achieved using the authenticated dictionary technique [10, 16, 17, 25]. The authenticated dictionary is a system for distributing data and supporting authenticated responses to queries about the data. The data originates at a secure central site (the repository) and is distributed to servers scattered around the network (responders). The responders answer queries about the data made by clients on behalf of the repository.

The roles or public keys whose delegated privileges are revoked are put on the repository of the revocation service by the resource owner. Before verifying the credential signatures in the VERIFY operation, the resource owner queries the revocation service to ensure that no public key whose delegated privileges are revoked appears on the delegation credential. Similarly, the revocation of affiliated role memberships can also be supported using a revocation service, which the verifier queries in the VERIFY operation to ensure the validity of the affiliated role memberships of intermediate delegators.

7. RELATED WORK

In Table 1, several properties of our delegation model and existing delegation models are compared. *Hier. Token* represents the hierarchical delegation protocol given in [11]. We denote with n the number of entities on a delegation chain. *Third-party* means whether the delegation chain verification algorithms require the participation of intermediate entities. *Cred. pass-down* refers to whether the delegation or role credential of the delegator has to be passed down to the delegatee. *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 the *RT* framework. The PolicyMaker [4] and KeyNote [3] trust management systems authorize decentralized access by checking a 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 [9, 13]. As noted earlier, KeyNote and SPKI/SDSI do not define explicit role abstractions and assume that all the certificates are available to the discovery algorithms.

There are several cascaded delegation [29] schemes for the proxy authentication and authorization, including nested signature schemes [30], delegation keys [26], and a combined approach [11]. These schemes do not support delegations to roles, and the delegation credentials are not as compact as ours. The security framework for Java-based computing environment in [29] uses roles in chained delegations to simplify the management of privileges. However, their delegations are made to individuals rather than to roles. Their term cascaded delegation has different meanings from ours, and refers to delegations where all the privileges of preceding entities on the chain are inherited by the delegatee.

8. CONCLUSIONS AND FUTURE WORK

We have proposed a role-based cascaded delegation model, which combines role-based decentralized trust management with cascaded delegation. The role-based cascaded delegation mechanism eliminates the need for distributed delegation chain discovery and its associated issues, because credentials are passed down to the delegatee at each delegation. The model provides a simple alternative for decentralized delegation.

We have also presented an implementation of role-based cascaded delegation using the Hierarchical Certificate-Based Encryption (HCBE) scheme. The use of HCBE in implementing role-based cascaded delegation achieves properties that cannot be realized in conventional credential systems. In particular, the use of HCBE scheme allows the authentication information of an arbitrarily long delegation chain to be captured by one short signature of constant size. The implementation also provides strong privacy protection for delegation participants. Issues such as the efficiency, scalability, security, privacy, and revocation have been discussed. We conclude that the role-based cascaded delegation protocol implemented by the HCBE scheme eliminates the delegation chain discovery problem and provides simple verification of delegation chains, without significant increase in the size of delegation credentials.

We are currently building an RBCD prototype in Java using the OASIS standard Extensible Access Control Markup Language (XACML) [31] as the policy language. XACML is a flexible XML specification for expressing access control policies. An implementation of XACML in Java [22, 31] has been released by Sun Microsystems. We also plan to investigate how our role-based cascaded delegation can be combined with existing role-based delegation mechanisms. It would be interesting to study how the distributed credential chain discovery algorithms in the *RT* framework [20] can be modified in order to work with role-based cascaded credentials.

Properties	Ours	*RT* framework [21]	KeyNote [3]	SPKI [9]	Hier. Token [11]
Cascaded	Yes	No	No	No	Yes
Storage	Distributed	Distributed	Centralized	Centralized	Distributed
Chain discovery	Not required	Required	Required	Required	Not required
Role-based	Yes	Yes	No	No	No
Cred. pass-down	Not required	N/A	N/A	N/A	Required
Third-party	Not required	Required	N/A	N/A	Not required
Num of sig.	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Privacy	Strong	Weak	Weak	Weak	Weak
Cryptographic op.	Pairing [2]	N/A	N/A	N/A	Exponentiation

Table 1: Comparisons of parameters in delegation systems that address the delegation chain issue.

9. ACKNOWLEDGEMENT

We would like to thank Seth Proctor at Sun Microsystems Lab for helpful discussions on XACML and its implementation.

10. REFERENCES

[1] 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.

[2] 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.

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

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

[5] 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*, pages 213–229. Springer, 2001.

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

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

[8] 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.

[9] 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):285–322, 2001.

[10] P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine. Authentic third-party data publication. In *Fourteenth IFIP 11.3 Conference on Database Security*, 2000.

[11] 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.

[12] FIPS 186-2 Digital signature standard, 2000.

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

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

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

[16] M. T. Goodrich, M. Shin, R. Tamassia, and W. H. Winsborough. Authenticated dictionaries for fresh attribute credentials. In *Proc. Trust Management Conference*, volume 2692 of *LNCS*, pages 332–347. Springer, 2003.

[17] M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen. Authenticated data structures for graph and geometric searching. In *Proc. RSA Conference—Cryptographers'Track*, pages 295–313. Springer, LNCS 2612, 2003.

[18] C. A. Gunter and T. Jim. Policy-directed certificate retrieval. *Software: Practice and Experience*, 30:1609–1640, September 2000.

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

[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*, pages 114–130, 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] M. Lorch, S. Proctor, R. Lepro, D. Kafura, and S. Shahi. First experiences using XACML for access control in distributed systems. In *Proceedings of the ACM Workshop on XML Security 2003*, pages 25–37, October 2003.

[23] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential aggregate signatures from trapdoor permutations. In *Advances in Cryptology – Eurocrypt '04*. Available at `http://eprint.iacr.org/2003/091/`.

[24] 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.

[25] M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proceedings of the 7th USENIX Security Symposium*, pages 217–228, 1998.

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

[27] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Commun. ACM*, 21:120–126, 1978.

[28] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29, Number 2:38–47, 1996.

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

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

[31] XACML. `http://www.oasis-open.org/committees/xacml/` and `http://sunxacml.sourceforge.net/`.

APPENDIX

A. HIERARCHICAL CERTIFICATE-BASED ENCRYPTION

The Hierarchical Certificate-based Encryption (HCBE) scheme by Gentry [15] 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 [5] 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$(s_i, info_{i+1})$: CA at i-th level runs this algorithm to certify the public key of the CA at level $i+1$ by computing a signature. The first input is the private key of CA_i, and the second input is a string $info_{i+1}$ that contains the public key $s_i\pi$ of the signer and the public key $s_{i+1}\pi$ of CA_{i+1}. The string $info_{i+1}$ may also include information such as the expiration date, etc. CA_i first computes a hash digest $H(info_{i+1})$, 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 $s_iH(info_{i+1})$ is given to CA_{i+1}.

Certification_of_Bob$(s_{n-1}, info_n)$: CA_{n-1} runs this algorithm to certify the public key of Bob. The first input is the private key of CA_{n-1}, and the second input is a string $info_n$ that contains the public key $s_{n-1}\pi$ of the signer and the public key $s_n\pi$ of Bob. CA_{n-1} computes the signature as $s_{n-1}H(info_n)$, which is the multiplication of CA_{n-1}'s private key s_{n-1} with the hash digest of the string $info_n$. The output signature is given to Bob.

Aggregation$(s_n, info', sig_2, \ldots, sig_n)$: This algorithm is run by Bob, who 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 s_n, the string $info'$ that contains the information of Bob, and a number of signatures [1] 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 $s_{i-1}H(info_i)$, for $2 \leq i \leq n$. Bob first computes a signature $s_nH(info')$ on the string $info'$. He then aggregates this signature with all the input signatures simply by adding them together, $S_{Agg} = s_nH(info') + \sum_{i=2}^{n} sig_i$. The output S_{Agg} is Bob's decryption key.

Encryption$(M, info_1, \ldots, info_n, info')$: Alice computes the ciphertext to send to Bob. The inputs are a message M, string $info_i$ of the certification at level i on Bob's chain for $1 \leq i \leq n$, and string $info'$ that Bob signs in **Aggregation** algorithm. 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 $g = \hat{e}(s_n\pi, H(info'))\Pi_{i=1}^{n-1}\hat{e}(s_i\pi, H(info_{i+1}))$, $\oplus$ denotes bit-wise XOR operation, H' is the other cryptographic hash function in the system parameters *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.

[1] The **Aggregation** algorithm can take any number of signatures.