On Improving the Performance of Role-Based Cascaded Delegation in Ubiquitous Computing

Danfeng Yao, Roberto Tamassia
Computer Science Department
Brown University
Providence, RI 02912
{dyao, rt}@cs.brown.edu

Seth Proctor
Sun Microsystems Laboratories
Burlington, MA 01803
Seth.Proctor@sun.com

Abstract

In ubiquitous computing environment, computing devices may have small storage units and limited bandwidths. A trust management system needs to be efficient in order to keep communication and computation costs low. The trust establishment mechanism needs to be flexible, because credentials are usually scattered at distributed *locations. Also, the authorization process needs to be decentralized and support dynamic resource-sharing in order to handle emergency situations. We discuss how to improve the efficiency, flexibility, and privacy of role-based cascaded delegations in a ubiquitous computing environment. Operations for managing delegation chains in the role-based cascaded delegation (RBCD) model are presented. These operations significantly improve the performance of the decentralized delegation in the RBCD model, without increasing the management overhead.*

Keywords: Cascaded delegation, Decentralized trust management, Ubiquitous computing

1 Introduction

Trust management systems provide access control in environments where initially unknown entities from different administrative domains interact and establish trust through mutually trusted parties. A number of trust management models have been proposed [4, 5, 7, 14, 15, 18, 20, 23, 24, 25]. Role-based decentralized trust models [18, 24] combine trust management with role-based access control (RBAC) [21]. In these models, privileges can be delegated to roles, and therefore delegation is efficient and scalable.

In a ubiquitous computing environment, computing devices may have small storage units and limited bandwidth. A trust management system needs to be efficient in order to keep communication and computation costs low. The trust establishment mechanism needs to be flexible because credentials are scattered at distributed locations. Most importantly, the authorization process needs to be decentralized and support dynamic resource-sharing in order to handle emergency situations. Role-based cascaded delegation (RBCD) [24] is a delegation model that facilitates large-scale dynamic resource-sharing in decentralized and pervasive environments, and is suitable for tasks where coalitions are dynamically formed by role members from different administrative domains. In this paper, we present several credential and trust management techniques that further improve the efficiency and flexibility of delegation transactions in the RBCD model for ubiquitous computing environments.

An interesting research topic is how to facilitate ubiquitous access while protecting the user's privacy. In particular, the physical presence of a user while accessing certain resources may be sensitive information that the user does not want to disclose. In this paper, we describe how cryptographic schemes can be utilized to protect the user's privacy in role-based access control models, such as RBCD, for ubiquitous computing environments. We describe the RBCD model next, and then summarize our contributions.

1.1 Role-Based Cascaded Delegation

The delegation-based access control model [4] supports decentralized authorization through delegations. The RBCD model [24] generalizes this concept to role-based access control. It supports decentralized role-based delegation by allowing individual role members to delegate privileges associated with their roles to others, without the participation of the administrator.

The RBCD protocol [24] comprises four operations: Initiate, Extend, Prove, and Verify [24]. The Initiate and Extend operations are used by a resource owner and an intermediate delegator, respectively, to delegate a privilege to a role. The PROVE operation is used by a requester to produce a proof of a delegation chain that connects the resource owner with the requester. The VERIFY operation decides whether the requester is granted access based on the proof. Delegation credentials in RBCD are accumulated at each delegation transaction and form a delegation chain. A delegation chain represents how trust or a delegated privilege is transferred from one role to another. It contains a sequence of delegation credentials that connects unknown entities and resource owners. For example, a user with role *manager* at Central Bank issues a delegation credential C_1 to role *clerk* at a state bank to authorize the right of accessing a document *doc*. A member of role *clerk* at the state bank delegates this right to a role at a county bank by issuing another delegation credential C_2. Credentials C_1 and C_2 form a delegation chain connecting the role at the county bank with the Central Bank. An efficient RBCD protocol [24] with compact delegation credentials can be realized based on an aggregate signature scheme [10].

1.2 Requirements in Ubiquitous Computing

A decentralized trust management system in a ubiquitous computing environment should efficiently handle dynamic access control situations, such as an emergency room (ER) scenario. Suppose that during an operation on a patient, doctors find that the help of experts from several other organizations are urgently needed. These experts need to be given temporary access to resources such as the medical records of the patient, in order for them to make medical decisions. Issuing authorizations through regular administrative channels may not be as fast as needed. The existing RBCD model is particularly suitable for this dynamic delegation scenario because doctors can issue delegations based on their credentials, without the administrator's participation. However, in RBCD, if a doctor wants to delegate privileges associated with several credentials (e.g., doctor role credential, ER access credential and medical consortium membership credential), he has to generate a separate delegation certificate for each of these credentials. Furthermore, the doctor has to repeat this delegation process for each of the roles that will receive the delegations. Therefore, delegation efficiency in RBCD needs to be further improved for a better performance in dynamic and emergency access control situations in a ubiquitous computing environment.

The role-based credential accumulation approach of RBCD avoids the need for the dynamic credential chain discovery, which may be potentially costly in terms of communication overhead. However, the ability to dynamically construct delegation chains is important for a flexible delegation model. This prompts us to investigate the possibility of integrating the credential chain discovery model with the RBCD model. Such an integrated model can support more flexible trust establishment than the RBCD model alone. However, there are several security and performance considerations for such an integrated model. A resource owner should be able to control the availability of its delegations to the credential discovery, because delegation chains constructed by discovery algorithms may give authorizations to a large number of unknown users. In addition, communication costs need to be considered in the credential chain discovery for better performance in ubiquitous computing. We extend the RBCD model [24] to include the support of credential chain discovery algorithms. The contributions of this paper are summarized next.

1.3 Our contributions

In this paper, we present several techniques to improve the efficiency and flexibility of role-based cascaded delegation in ubiquitous computing environments. Our contributions are summarized as follows.

1. We support the use of delegation predicates and constraints in the RBCD language model. Using these specifications, the scope of delegation recipients can be flexibly adjusted. In Section 2, we describe our language model, which supports predicates and constraints. We describe in Section 4 how constraints are specified in a delegation transaction and the algorithm for verifying a delegation chain with constraints. Delegation constraints in decentralized role-based trust management systems such as RBCD can fine-tune the delegation scope. We show in Section 6 that delegation constraints improve the security of resources during the credential chain discovery by specifying how a delegated privilege is propagated.

2. In ubiquitous computing environments, delegations usually take place on small devices that have low computation power. To improve the delegation efficiency of RBCD, we present a method that reduces the workload of a delegator in the case where there are multiple recipient roles and multiple delegation chains to extend. In Section 5, we describe a new operation of the RBCD protocol that reduces the number of credentials generated by a delegator from $n \times m$ to $n + m$, where n delegation chains are to be extended to m roles. This is achieved through the use of local roles. An intermediate delegator only needs to perform n EXTEND operations and m INITIATE operations, as opposed to $n \times m$ EXTEND operations in the original RBCD protocol [24]. Merged credentials can be easily *split* back into individual credential chains. This flexibility of manipulating credentials allows an intermediate delegator to extend multiple delegation chains or a portion

of merged privileges.

3. We show in Section 6 how to improve the flexibility of the trust establishment mechanism in RBCD, by dynamically constructing longer delegation chains from RBCD credentials distributed across the network. We present the *combined RBCD* (cRBCD) protocol to efficiently retrieve distributed credentials. The cRBCD model has several advantages. The use of RBCD credentials partially reduces communication and computation costs, because portions of delegation chain are already captured in RBCD delegation credentials. The credential chain discovery allows the credential holders to establish trust with a larger number of resources through the discovery. To achieve a tighter security and better performance, we also modify the existing credential chain discovery algorithm. Our cRBCD model improves the protection of shared resources by allowing resource owners to specify attribute constraints that fine-tune the scope of delegations. The resource owners are given more control power over how discovered delegation credentials are used for the trust establishment of shared resources.

We also discuss in Section 7 how to utilize cryptographic schemes to protect the user's privacy in role-based access control models for ubiquitous computing environments.

1.4 Organization of the paper

The rest of this paper is organized as follows. Definitions and our language model are given in Section 2. In Section 3, we give an example to illustrate the decentralized authorization feature of RBCD in an ubiquitous computing environment. In Section 4, we describe how attribute constraints of a delegation chain are specified and verified in RBCD, which is used in the following sections. Then, the *merge* RBCD (mRBCD) protocol is presented in Section 5. We describe how the credential chain discovery is combined with RBCD in Section 6. Section 7 discusses how the user privacy of RBAC models including the role-based cascaded delegation model can be protected using existing anonymous authentication techniques. The related work and conclusions are given in Section 8 and 9, respectively.

2 Terminology and language model

As in the original RBCD model [24], 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 credentials. An *affiliated role credential* is the credential for an affiliated role, and is signed by the administrator of the role. A *delegation credential* is the credential for proving a delegated privilege. A *privilege* can be role assignment(s) or action(s) on a resource. An *extension credential* or *extension certificate* is generated and signed by a delegator on delegation transaction information, such as the identities of the delegator and delegatee, and the delegated privilege. A *complete delegation credential* includes a signature from a requester, extension credentials, and role credentials. It authenticates a delegation chain of a privilege connecting a resource owner with a requester. 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 information of the requester. In this paper, we use curly brackets (for example $\{C\}$) to refer to a set of RBCD delegation credentials.

2.1 Language model

Role r administered by entity A is denoted as $A.r$, as in the original RBCD model [24]. 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 $D \xrightarrow{A.r} B.s$. The above delegation means that entity A delegates to role $B.s$ privileges that are associated with role $A.r$ *and* are controlled by entity A. It does not automatically grant $B.s$ the privileges that are *delegated* to role $A.r$. Any member 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 $E \xrightarrow{A.r} C.t$.

We introduce attribute constraints to role and delegation credentials. Attribute constraints are assigned in role credentials by role administrators, or in delegation credentials by delegators. For example, a resource owner specifies that the number of times a delegation privilege can be further extended is at most two. We express attribute constraints on the arrow of a delegation expression, e.g., $A \xrightarrow{A.r\ [\mathsf{attr}=v]*} B.s$, where * is a wildcard representing that there are zero or more attribute constraints following $A.r$.

In our combined RBCD (cRBCD) model, we define boolean attribute *isPropagatable* and *all* for delegation transactions, e.g., $A \xrightarrow{A.r\ [\mathsf{isPropagatable}=v, \mathsf{all}=v']} B.s.$

- *isPropagatable*: a boolean attribute assigned by a resource owner to specify whether or not the delegated privilege (e.g., $A.r$) also applies to those who are *delegated* the recipient role (e.g., $B.s$). If *isPropagatable* is true, those who are *delegated* the recipient role (e.g., $B.s$) are also entitled to the privilege (e.g., $A.r$). For example, if $A \xrightarrow{A.r\ [isPropagatable=true]} B.s$, and $B \xrightarrow{B.s} C.t$, then members of role $C.t$ are also members of $A.r$.

- *all*: a boolean attribute assigned by a resource owner to specify whether or not the delegated privilege (e.g., $A.r$) also includes all the privileges that are delegated to role $A.r$. If *all* is set to true, then all the delegations associated with role $A.r$ are also given to role $B.s$. For example, if $E \xrightarrow{E.p} A.r$ and $A \xrightarrow{A.r\ [all=true]} B.s$, then members of $B.s$ are also members of $E.p$.

isPropagatable and *all* define whether discovered RBCD credentials can be accepted by a resource owner. If a sequence of discovered delegation credentials form a delegation chain connecting a resource owner with a requester, then for this delegation chain to be accepted by the resource owner, *isPropagatable* in the delegation certificate issued by the resource owner must be true. In addition, the rest of delegeation credentials must have *all* set to true. Note that a credential with *isPropagatable* being false can still be extended by running EXTEND algorithm. We discuss the security implications brought by these attributes in Section 7.

We introduce predicates for our language model to express requirements for attributes of a delegatee or a delegation chain. Each predicate restricts the scope of delegation recipients. For example, the ranking of a delegatee's role should be greater than or equal to *senior engineer*. We express a predicate as $P(attr, v)$, where *attr* is the name of an attribute of delegation recipients, v is a value, and P is an evaluation function such as greater-than and less-than. A predicate is evaluated to either true or false. It may specify a constraint based on a literal value, as in $> (attr, 3)$, or on the value of another attribute, as in $> (attr_1, attr_2)$. All predicates in a credential must be satisfied in order for the credential to be valid. Predicates are written on the right of delegation expression as in $A \xrightarrow{A.r} B.s\ [P(attr, v)*]$, where * is a wildcard and represents that there is zero or more predicates following $B.s$. Delegation predicates and constraints are used throughout this paper. In Section 4, we define how attribute constraints and predicates of a delegation chain are specified and verified in RBCD.

3 Example scenarios

In this section, we give a dynamic resource-sharing example to illustrate the decentralized delegation feature of RBCD in ubiquitous computing.

In an ubiquitous computing environment, the access to the medical refrigerator at a hospital L is controlled. Bob is a doctor at a hospital L, and has the access to the refrigerator. He has the role certificate issued by L stored in his hand-held device.

$$L \xrightarrow{L.doctor} \text{Bob} \quad (1)$$

Bob and his co-workers are joined by their collaborators from a medical center H in an emergency operation on a patient. The collaborators are members of role *poison expert* at H, and they carry smart cards that store their digital credentials including their role credentials, for example:

$$H \xrightarrow{H.poison\_expert} \text{Adam} \quad (2)$$

The operation requires the experts to access the medical refrigerator in the clinic. In such an emergency situation, authorizations have to be issued fast, which may not be accomplished through the regular administrative channel. RBCD supports decentralized role-based delegation by allowing individual role members to delegate privileges associated with their roles to others, without the participation of the administrator. Therefore, the authorization can be issued much faster. Bob uses his hand-held device to create a delegation credential (3) that includes his role credential (1) and a delegation extension certificate signed with his private key.

$$\begin{array}{lcl} L & \xrightarrow{L.doctor} & \text{Bob} \\ Bob & \xrightarrow{\text{open the refrigerator}} & H.poison\_expert \end{array} \quad (3)$$

In credential (3), the privilege of opening the refrigerator is delegated by Bob to members of role $H.poison\_expert$. Bob's role credential shows that he is allowed to delegate privileges associated with doctors. This credential (3) is submitted to the credential server of L. When Adam, a member of $H.poison\_expert$, wants to access the refrigerator, he puts his smart card containing his role credential (2) in a card reader. His role credential and the credential (3) retrieved from the credential server of L are verified, and Adam's access is granted.

This role-based authorization is established fast, because it does not involve the role administrators of either organization. It is scalable because of the role abstraction. And aggregate signature [10] and short signature [11] schemes can be used to reduce the credential size to improve the transmission efficiency.

If the role $L.doctor$ receives delegations of other roles from different organizations, Bob can extend these delegations to his collaborators, too. In the case where there are multiple collaborative roles and multiple delegations to extend, the MERGE EXTEND operation in our proposed mRBCD protocol can significantly reduce the computation overhead of the delegator.

In a ubiquitous computing environment, delegation credentials are scattered across the network at distributed storage locations. We improve the existing credential chain discovery algorithm [19] and present a cRBCD protocol to efficiently retrieve distributed credentials. cRBCD also allows resource owners to specify how their delegations are extended, by using delegation predicates and constraints.

In our models, the public-key of the recipient's role administrator is needed for identifying the role in a delegation certificate. For example, to issue credential (3), Bob needs the public-key of the role administrator in Hospital H. We assume that Bob obtains the public-key from a public directory, and verifies the validity and authenticity of the key.

In the following sections, we first define algorithms for specifying and verifying predicates and constraints in role-based cascaded delegation, then mRBCD protocol and cRBCD protocol are presented, respectively.

4 Predicates and constraints in RBCD

In this section, we describe the operations that specify and verify the attribute constraints and predicates of a delegation chain, which are used to fine-tune the scope of a delegation chain. We will see in Section 6 the impact of delegation constraints on the propagation of a delegated permission and as a consequence on the number of delegatees in RBCD.

Predicates and constraints are specified on the delegation certification issued by a delegator at the delegation transaction. For example, the resource owner specifies that only members of role *doctor* with a ranking at least $D2$ at a hospital can access a medical database. Role attributes are also specified in affiliated role credentials by role administrators. This predicate has two impacts to the delegation: only those role members who satisfy this predicate (1) can access the database, and (2) can generate *valid* delegation extensions of this privilege to others. The latter means that delegation credentials issued by those who have lower rankings are *invalid* and cannot be accepted by resource owners at the verification. For a delegation chain to be valid, all the predicates and constraints have to be satisfied at verification. Because of the decentralized nature of the delegation model, role members with lower ranks are not prevented from issuing delegation extensions. However, at the verification, delegation chains that contain credentials issued by these unauthorized role members will not be accepted by a resource owner, and thus the delegation constraints and predicates are enforced.

In our protocol definitions, we use *Pred* to denote a predicate, and *Cons* to denote an constraint. All specified predicates with respect to every delegation transaction must be satisfied in order for the delegation chain to be valid.

Our role-based cascaded delegation protocol supporting constraints has four operations: INITIATE, EXTEND, PROVE, and VERIFY. We define them below.

- INITIATE(P_{D_0}, s_{D_0}, $D_0.priv$, $A_1.r_1$, P_{A_1}, $[\mathit{Cons}]*$, $[\mathit{Pred}]*$): 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 $D_0.priv$. It is similar to INITIATE operation of the original RBCD protocol in [24]. The difference is that here a delegator is allowed to specify predicates and constraints to refine delegation scope in the credential.

  Inputs include public key P_{D_0} of entity D_0, corresponding private key s_{D_0}, delegated privilege $D_0.priv$, role name $A_1.r_1$, public key P_{A_1} of role administrator A_1, and zero or more constraints $[\mathit{Cons}]*$ for the transaction and predicates $[\mathit{Pred}]*$ for delegatees. Output is a partial delegation credential C_1 for the role $A_1.r_1$, represented as

  $$D_0 \xrightarrow{D_0.priv\ [\mathit{Cons}]*} A_1.r_1[\mathit{Pred}]* .$$

  The statement of C_1 has the information about the delegation transaction, including public key P_{D_0}, privilege $D_0.priv$, information about role $A_1.r_1$ such as name and public key of administrator A_1, and delegation constraints and predicates. The certificate is signed using private key s_{D_0}.

  This operation is also used by a role administrator A_1 to generate an affiliated role certificate for a member D_1, if the second to the last argument is a public key of D_1. The role credential is expressed as follows, where [*Cons*]* is where role attributes of an individual are specified in the affiliated role certificate.

  $$D_0 \xrightarrow{D_0.priv\ [\mathit{Cons}]*} D_1[\mathit{Pred}]* .$$

- EXTEND(s_{D_n}, $D_0.priv$, C_n, R_{D_n}, $A_{n+1}.r_{n+1}$, $P_{A_{n+1}}$, $[\mathit{Cons}]*$, $[\mathit{Pred}]*$):

  This operation is run by an intermediate delegator D_n to extend delegated privilege $D_0.priv$ to role $A_{n+1}.r_{n+1}$. In role-based cascaded delegation, entity D_n needs to prove to be a member of a role A_n that has already been delegated $D_0.priv$. This operation differs from the EXTEND of the original RBCD model [24] in that the credential C_{n+1} issued by the delegator D_n needs to prove the satisfications of all the predicates associated with D_n's role $A_n.r_n$ in C_n.

  Inputs to this operation are private key s_{D_n} of delegator D_n, delegated privilege $D_0.priv$, partial delegation credential C_n that gives privilege $D_0.priv$ to role $A_n.r_n$, role credential R_{D_n} of delegator D_n, role name $A_{n+1}.r_{n+1}$, public key $P_{A_{n+1}}$ of role administrator A_{n+1}, and constraints $[\mathit{Cons}]*$ and predicates $[\mathit{Pred}]*$ specified by delegator D_n. Credential C_n is retrieved from a credential server. The partial delegation credential C_n is a function of the preceding extension credential and role credentials with predicates and constraints.

An extension credential denoted by $(D_n \xrightarrow{D_0.priv\ [\mathit{Cons}_n]*} A_{n+1}.r_{n+1}\ [\mathit{Pred}_n]*)$ is generated as an intermediate product of operation EXTEND. Its statement contains information about delegated privilege $D_0.priv$, role $A_{n+1}.r_{n+1}$, and predicates and attribute specifications. It is signed with 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 $(A_n \xrightarrow{A_n.r_n\ [\mathit{Cons}'_n]*} D_n\ [\mathit{Pred}_n]*)$, and the extension credential described above.

Credential C_{n+1} may simply be the delegation credential C_n together with the two individual credentials. Alternatively, D_n can compute a delegation credential for the role $A_{n+1}.r_{n+1}$ as in the original role-based cascaded delegation implementation [24] using aggregate signatures [10]. Credential C_{n+1} is placed on a credential server.

- PROVE$(s_{D_n}, D_0.priv, R_{D_n}, C_n)$:

  THis operation is performed by a requester D_n who wants to exercise privilege $D_0.priv$. D_n is an affiliated member of role $A_n.r_n$. 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 public key P_{D_n} that appears on role credential R_{D_n} of the requester. R_{D_n} and partial delegation credential C_n together are to prove that D_n is authorized privilege $D_0.priv$.

- VERIFY(F):

  This operation is performed by resource owner D_0 to verify that proof F produced by requester D_n correctly authenticates the delegation chain of privilege $D_0.priv$. It is similar to VERIFY in [24], but needs extra work in order to check all the predicates in credentials are satisfied. This is done by evaluating predicates with values of attributes, which may come from constraint statements $[\mathit{Cons}]$ specified on role credentials in F. Specifically, the verifier checks whether the signatures in F correctly authenticates the delegation chain. This includes the authentication of each delegation extension $(D_{i-1} \xrightarrow{D_0.priv\ [\mathit{Cons}_{i-1}]*} A_i.r_i\ [\mathit{Pred}_{i-1}]*)$, and entity D_i's affiliated role membership $(A_i \xrightarrow{A_i.r_i\ [\mathit{Cons}'_i]*} D_i\ [\mathit{Pred}'_i]*)$, 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.

In general, delegation constraints and predicates are for issuing fine-grained delegation credentials. We show in Section 6 an interesting application of constraints where they are used to control how credential chain discovery propagates delegated privileges and restrict the scope of resource-sharing.

Note that there are two types of delegation constraints depending on their semantics. One type of constraints should be enforced throughout a delegation chain. For example, the constraint on the length of a delegation chain. The other type is effective with respect to an individual delegation transaction, rather than the entire chain. For example, the role ranking of a delegatee. Distinguishing them depends on the semantics of attributes defined by applications, and is out of the scope of this paper.

Security The RBCD protocols in this and the two subsequent sections can be implemented using any valid signature schemes. The security of the protocols is based on the unforgeability of signature scheme used to sign credentials. We allow adversaries to observe the traffic on the network. They can also participate delegation processes, that is, an adversary may be a valid member of a role who can issue delegations and submit access requests. In our RBCD protocols, an adversary cannot successfully submit access requests on behalf of others, in particular, in the name of other role members. Similarly, an adversary cannot successfully issue delegations on behalf of others, even if she obtains another entity E's role credential and delegation credentials issued to E. This is because an adversary cannot forge a valid signature of E. Our protocols are secure against replay attacks, because it is infeasible for an adversary to forge a valid signature on a verifier-chosen nonce signed with other member's private key. These imply that an adversary cannot obtain or delegate privileges that are not authorized to them. As described in [24], the RBCD protocol implemented with aggregate signatures [10] has a compact representation of credentials. In the next section, we present a method that can effectively reduces the number of delegation credentials to be issued.

5 Merging delegation chains

In ubiquitous computing, delegators may use small devices that have low computation power. To improve the delegation efficiency of RBCD, we present a method that reduces the workload of a delegator in the case where there are multiple recipient roles and multiple delegation chains to extend. We introduce a new operation MERGE EXTEND which is run by a delegator D as follows. D first creates a new local role, for example, D.*local*, and extends each of his delegation credentials to role D.*local*. This gives a credential set $\{C\}$. Then D delegates role D.*local* to delegatees, by initiating a new delegation chain. Set $\{C\}$ and the credential for the new delegation chain are issued to the delegatees. We refer this role-based delegation protocol as the *merge* RBCD (mRBCD) protocol. Details of the mRBCD protocol with MERGE EXTEND operation are defined next.

5.1 mRBCD protocol

The mRBCD protocol has five operations: INITIATE, EXTEND, MERGE EXTEND, PROVE, and VERIFY. Operation MERGE EXTEND is the new operation that handles the *merging* of delegation chains. All operations support the use of delegation constraints and predicates.

- INITIATE(P_{D_0}, s_{D_0}, $D_0.priv$, $A_1.r_1$, P_{A_1}, [*Cons*]$*$, [*Pred*]$*$):

  This operation is the same as in Section 4.

- EXTEND(s_{D_n}, $D_0.priv$, C_n, R_{D_n}, $A_{n+1}.r_{n+1}$, $P_{A_{n+1}}$, [*Cons*]$*$, [*Pred*]$*$):

  This operation is the same as in Section 4.

- MERGE EXTEND(s_D, R_D, $\{C\}$, $A.r$, P_A, [*Cons*]$*$, [*Pred*]$*$):

  This algorithm is run by an entity D to delegate privileges associated with a set of credentials $\{C\}$ to role $A.r$. The inputs include private key s_D of entity D, a role credential R_D of D, a set of credentials $\{C\}$ that are issued to the role of D, a delegation recipients' role $A.r$, public key P_A of role administrator A of role $A.r$, and constraints [*Cons*]$*$ and predicates [*Pred*]$*$ of attributes.

  Delegator D first locally creates a new role D.*local*. For each credential $C_j \in \{C\}$, D extends the delegated privilege $priv_j$ in C_j to the new role D.*local* by running EXTEND(s_D, $priv_j$, C_j, R_D, $A.r$, P_A, null, null) method above. (Two null arguments represent that no constraints and predicates are defined.) EXTEND returns a credential C'_j that delegates privilege $priv_j$ to role D.*local*. The set of credentials returned by EXTEND operations is denoted as $\{C'\}$. Next, delegator D initiates an intermediate delegation chain that delegates role D.*local* to role $A.r$, by running INITIATE(P_D, s_D, D.*local*, $A.r$, P_A, [*Cons*]$*$, [*Pred*]$*$) algorithm to generate a delegation credential C_r. C_r corresponds to a delegation chain for role D.*local*, which is called an *intermediate delegation chain*. The outputs of this operation are credential set $\{C\}$ and credential C_r, which together are delegation credentials for members of role $A.r$.

- PROVE(s_E, $V.priv$, R_E, $\{C'\}$):

  This algorithm is run by a requester E with role credential R_E, who wants to access the privilege $V.priv$ controlled by a resource owner V. Its main difference from PROVE of the original RBCD protocol [24] is that a proof produced by this algorithm may be constructed from two or more delegation chains, as a result of MERGE EXTEND operations.

  Inputs to this operation are private key s_E of entity E, role credential R_E, and a set of delegation credentials $\{C'\}$ that collectively form a delegation path between the resource owner and the requester E. $\{C'\}$ contains one delegation chain initiated by resource owner, and zero or more intermediate delegation chains initiated by intermediate delegators [1]. The operation produces a proof F that proves: (1) the role of E is delegated privilege $V.priv$, (2) E is a valid role member, and (3) E can authenticate his public key.

- VERIFY(F):

  This operation is performed by a resource owner V on proof F, which is submitted by a requester. If proof F does not involve any local roles, i.e. none of delegation certificates in F is generated by MERGE EXTEND, then VERIFY algorithm of the original RBCD protocol [24] is called. Otherwise, V authenticates proof F as follows.

  V first verifies delegation credentials $C_1, \ldots, C_n$ in F, each of them representing a delegation chain. Role and extension credentials that constitute each delegation chain are verified, and the satisfactions of all delegation predicates are checked. Then, the correct *linkages* between delegation chains are verified. That is, F should contain the proof of the following: (1) credential C_1 is issued by resource owner V, delegated privilege in C_1 is the requested privilege, and credential C_n is issued to requester E's role; (2) for $i \in [2, n]$, the delegated privilege in C_i is the last role (a local role) that receives delegation in credential C_{i-1}; (3) for C_i where $i \in [2, n]$, the original delegator is a valid member of the second to last role that receives delegation in credential C_{i-1} (for example in Figure 1, Alice is a member of A.*doctor*). Finally, the proof F also allows the resource owner to verify the role membership and the public key of the requester.

[1]The number of intermediate delegation chains equals the number of times operation MERGE EXTEND is run.

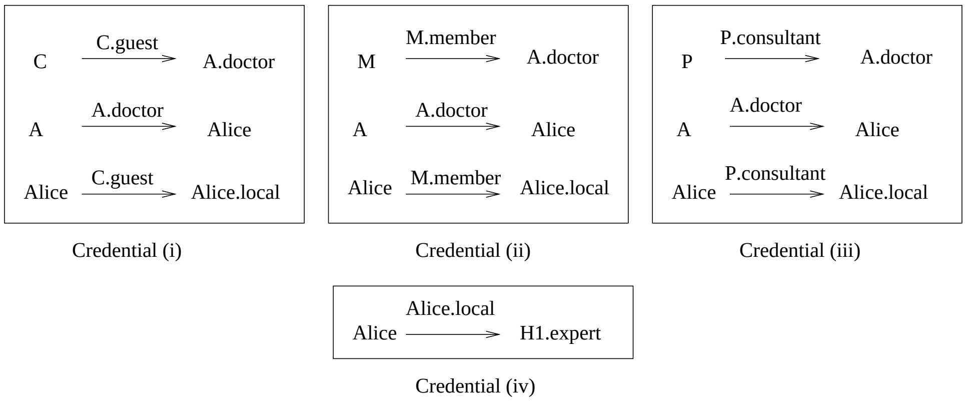

Figure 1. An example of a delegation credential generated by the MERGE EXTEND**. Alice is a member of role** A*.doctor* **at hospital** A**. The role** A*.doctor* **is delegated the role** C*.guest***,** M*.member***, and** P*.consultant* **by organization** C**,** M**, and** P**, respectively. Alice extends these three delegations to role** H_1*.expert* **at hospital** H_1**, through a local role** *Alice.local* **that she creates. The local role is delegated to role** H_1*.expert***. The picture shows the components of the delegation credential received by role** H_1*.expert***.**

An example of a delegation credential generated by the MERGE EXTEND operation is shown in Figure 1.

5.2 Efficiency Improvement

The MERGE EXTEND operation reduces the workload of a delegator in the case where n delegation chains are to be extended to m roles. The number of credentials needed to be generated by the delegator is reduced from $n \times m$ to $n + m$. An intermediate delegator only needs to perform n EXTEND operations and m INITIATE operations, as opposed to $n \times m$ EXTEND operations in the original RBCD protocol [24]. This improvement significantly saves the computation time of intermediate delegators that are typically individual users in our model. The mRBCD protocol can efficiently issue authorizations to a large number of entities without the involvement of administrators. This is important for forming large-scale dynamic collaborations in emergency medical situations.

The MERGE EXTEND operation does not introduce any management overhead. It can be implemented using data structures such as a linked list. Each RBCD credential is a linked list of individual certificates. MERGE EXTEND operation first links multiple linked lists that correspond to multiple RBCD credentials, and then extends the resulting linked list by appending new certificates from the intermediate delegator. Merged credential can also be easily reconstructed into individual delegation credentials, which is useful when the holder of the credential only wants to extend part of the delegated privileges to others.

Next, we describe an approach that improves the flexibility of role-based cascaded delegation in distributed environment.

6 Credential chain discovery in RBCD

The role-based credential accumulation approach of RBCD avoids the need for the dynamic credential chain discovery [19], which may be potentially costly in terms of the communication overhead. However, the ability to construct dynamic delegation chain is important for a decentralized delegation model. This is particularly crucial in a ubiquitous computing environment, where the trust establishment mechanism among unknown entities takes places frequently and thus needs to be flexible. In order to improve the flexibility of the RBCD model, we propose an integrated approach that combines credential chain discovery with the RBCD protocol. When performing a verification, the credential chain discovery algorithm is used to collect RBCD credentials to construct a longer delegation path. Through the longer delegation path, the trust between the requester and resource owner can be established, while this may not be possible through individual RBCD credentials alone. An example is shown in Figure 2. Compared to constructing delegation chains from single credentials as in the existing credential discovery framework [19], using RBCD credentials as building blocks can greatly reduce the costs associated with the discovery process. We call this integrated model as the *combined* RBCD (cRBCD).

6.1 Discovery algorithm

We have modified the credential chain discovery algorithm by Li, Mitchell, and Winsborough [19] to improve the performance of discovery process and the security of shared resources. First, we do not require credential servers of intermediate delegators to relay credential query results from one server to another. They only need to response to creden-

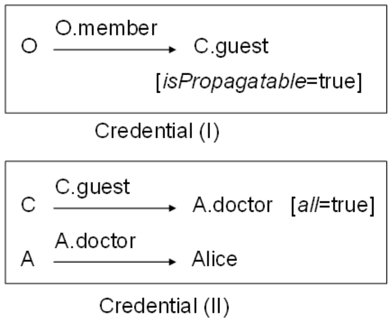

Figure 2. A delegation chain consists of Alice's credential (II) and credential (I), which is retrieved by the credential chain discovery. (I) states that a member of $C.guest$ is also a member of $O.member$. Because *isPropagatable* is true in credential (I) and *all* is true in credential (II), the delegated privilege $O.member$ can be propagated to those who are delegated role $C.guest$. Alice is delegated role $C.guest$ in credential (II), where *all* indicates privileges delegated to $C.guest$ are also authorized to $A.doctor$. Therefore, Alice is authorized the role $O.member$.

tial queries submitted by a requester or a resource owner. Second, an original delegator is able to specify *isPropagatable* and *all* attributes to control how delegated privileges are used in constructing valid delegation chains. These attributes improve the delegation granularity of the discovery algorithm, and can effectively control the scope of delegation recipients. We further explore the security implications of this constraint in Section 7.

Next, we give a breath-first delegation chain discovery algorithm that is run by a requester E. It takes an initial set of E's credentials $\{C\}$ and a target privilege $V.priv$, and outputs a set of delegation credentials $\{C'\}$ that collectively authorizes $V.priv$ to the requester E by resource owner V. The algorithm assumes that credentials are stored by delegation receivers (or their credential servers). Let X be a global set X for storing discovered credentials.
Discover$(\{C\}, V.priv)$:

1. For each delegation credential C_i in $\{C\}$, let the *original delegator* be O_i and the delegated privilege be role $O_i.r_i$. Requester E queries the credential server of delegator O_i to retrieve any credential at O_i that are issued to role $O_i.r_i$.

2. If nothing is retrieved by any of the O_i's servers, then the algorithm returns an empty set since there are no valid delegation chains connecting E's roles.

3. Otherwise, E adds delegation credentials $\{C\}$ to set X. For each delegation credential returned by the O_i's servers, E checks if the delegated privilege is $V.priv$.

    - If yes, then a delegation chain is successfully discovered. Denote the credential that delegates privilege $V.priv$ as C_0. From C_0 and credentials in X, E constructs a delegation chain connecting himself and the resource owner V. This can be done by iteratively finding the delegation credential $C_i \in X$ whose delegated privilege is the same as the last role that receives the delegation in credential C_{i-1}. The iteration terminates when the last role that receives the delegation in C_i is one of the requester E's. Credentials in the delegation chain are returned.

    - If there is no credential returned by O_i's server delegates the privilege $V.priv$, requester E resets $\{C\}$ to be the delegation credentials retrieved from O_i's servers, and repeats the discovery process by running Discover$(\{C\}, V.priv)$.

The algorithm can also be run by the resource owner V. The algorithm run by a requester is likely to perform better, because it processes a smaller number of credentials. Next, we present the cRBCD protocol that makes use of the discovery algorithm.

6.2 cRBCD protocol

The cRBCD protocol has four operations: INITIATE, EXTEND, PROVE, and VERIFY. The MERGE EXTEND operation of mRBCD in Section 5 can also be added to cRBCD to support the merging of delegation credentials, and is not repeated here. In the cRBCD protocol, an original delegator specifies boolean attribute *isPropagatable* and *all* in operation INITIATE, and the credential chain discovery algorithm is used to dynamically construct delegation chains in operation PROVE. The cRBCD protocol is defined as follows.

- INITIATE(P_{D_0}, s_{D_0}, $D_0.priv$, $A_1.r_1$, P_{A_1}, [*Cons*]$*$, [*Pred*]$*$):

  This operation is run by a resource owner D_0 to initiate a delegation chain. If the delegated privilege $D_0.priv$ is authorized not only to the affiliated members of $A_1.r_1$ but also to the delegated members of $A_1.r_1$, then attribute *isPropagatable* is set to true. If all the privileges that are *delegated* to role $D_0.priv$ are also delegated to role $A_1.r_1$, then set *all* to true.

- EXTEND(s_{D_n}, $D_0.priv$, C_n, R_{D_n}, $A_{n+1}.r_{n+1}$, $P_{A_{n+1}}$, [*Cons*]$*$, [*Pred*]$*$):

  This operation is the same as EXTEND operation in RBCD protocol of Section 4.

- PROVE($s_E, V.priv, R_E, \{C\}$):

  This algorithm is run by requester E for proving that he is delegated privilege $V.priv$, where V is a resource owner. The inputs are private key s_E of entity E, privilege $V.priv$, a role credential R_E of E, and a set of delegation credentials $\{C\}$ issued to E's role. For each credential in $\{C\}$, check if any of them delegates privilege $V.priv$. If yes, construct and output a proof F by running $\text{PROVE}(s_E, V.priv, R_E, \{C\})$ of mRBCD protocol in Section 5. Otherwise, run algorithm Discover$(\{C\}, V.priv)$. If Discover returns an empty set, then no valid delegation chain can be found and an empty set is returned. If Discover returns a non-empty credential set $\{C'\}$, then output a proof F that contains $\{C'\}$, role credential R_E, and a signature signed with s_E on a nonce chosen by the verifier.

- $\text{VERIFY}(F)$:

  Let $C_1, \ldots, C_n$ be delegation credentials in proof F, each of them representing a delegation chain. Delegation credential C_1 is issued by resource owner V, the delegated privilege in C_1 is the requested privilege. The delegation credential C_n is issued to E's role.

  Attribute *isPropagatable* must be true in credential C_1, and attribute *all* must be true in all of credential $C_2, \ldots, C_{n-1}$. Otherwise, the verification fails. V then authenticates the delegation credential C_1 through C_n by verifying signatures and evaluating predicates. Then, the correct *linkages* between delegation chains are checked: delegated privilege in C_i (for $2 \leq i \leq n$) is the same as the last role receiving delegation in credential C_{i-1}. For example in Figure 2, C.*guest* is the last role on the delegation chain in credential (I). It is also the delegated privilege in credential (II). Finally, the proof F also allows the resource owner to verify the role membership and the signature of the requester.

The credential chain discovery algorithm retrieves independently issued credentials and dynamically constructs delegation chains. It also helps to reduce the number of credentials carried by delegatees. Next, we discuss several features of our cRBCD protocol.

6.3 Flexibility and Security Improvements

The credential chain discovery in combination with the cascaded delegation allows the credential holders to establish trust with a larger number of resources through the discovery. This flexibility does not cause any significant management overhead because the credential infrastructure is the same as in the RBCD model [24] and the discovery process can be fully automated. The use of RBCD credentials reduces communication and transmission costs in the credential chain discovery. It requires fewer queries to construct a delegation chain from RBCD credentials than from individual credentials, because some portion of a delegation chain is already captured in an RBCD credential, and does not need to be discovered from scratch.

In decentralized delegation systems, protection of resource security relies crucially on the trustworthiness of delegators. In the cRBCD model, this is particularly important because delegation chains are constructed *ad-hoc*. The credential chain discovery is used to give more possibilities for an unknown requester to establish trust with a resource owner. In the mean time, our cRBCD model provides resource owners a way of restricting the scope of resource-sharing by specifying the boolean attribute *isPropagatable* and *all*.

We improve the delegation granularity by making two distinctions. Attribute *isPropagatable* distinguishes the affiliated members of a recipient role from the delegated members of the role. The existing discovery algorithm [19] implicitly assumes a delegation can be applied to both affiliated members and delegated members of a recipient role. This may give authorizations to a large number of users, which may not be desirable by the resource owner in some applications. We give resource owners the ability to fine-tune the scope of recipient roles in order to improve the protection of shared resources.

Another distinction is made on the scope of the delegated privilege (e.g., $A.r$) using attribute *all*. A resource owner can restrict that only permissions associated with $A.r$ defined in organization A are delegated. Otherwise, when *all* is true, the delegated privilege also includes all the permissions that are delegated to role $A.r$ by other organizations denoted by *Org*. Only in the latter case, the delegation can be used to construct a discovered delegation chain, which gives more authorizations to the recipient role than in the former case. The existing discovery algorithm [19] implicitly assumes the latter case, which may not always be necessary and could compromise the security of the shared resources owned by other organizations Org. In comparison, we allow resource owners to control over the scope of the delegated privileges. The decision made by a resource owner can be based on factors such as the trustworthiness of recipients, the scope of recipients, and the sensitivity of the to-be-shared resources belong to other organizations Org. In summary, the cRBCD model improves the granularity of decentralized delegations, which strengthens the protection of shared resources.

7 User Privacy in Ubiquitous Computing

An important privacy consideration in ubiquitous computing is that the *smart* computing environment may become an intelligent surveillance system that monitors every move of its users. One of the challenges for decentralized trust management models in a ubiquitous computing environment is how to protect the user privacy while maintaining the security of the shared resources. In this section, we describe how existing group signature schemes can be used to protect the *identity* of a delegator in role-based delegation models for ubiquitous computing environments. The goal is to allow an individual to delegate on behalf of her role without disclosing her identity.

In the existing role-based decentralized trust management models including the ones in the previous sections, the role membership of a user is authenticated by a role certificate issued by the role administrator. The certificate usually contains the public-key or the identity of the user. However, a key observation is that in the role-based models what is important to authentication is the role membership of a user, not his identity. Therefore, a user only needs to prove to a resource owner his role membership without disclosing his identity. The resource owner only needs to verify the validity of a role membership, not the identity of a requester. In the mean time, for accountability reasons, there should be a mechanism to revoke the anonymity of the user and reveal his identity.

The anonymous role-based authentication can be achieved using group signature schemes. Group signatures, introduced by Chaum and van Heijst [13], allow members of a group to sign messages anonymously on behalf of the group. Only a designated group manager is able to identify the group member who issued a given signature. Furthermore, it is computational hard to decide whether two different signatures are issued by the same member. For the role membership authentication in role-based models such as RBCD, the resource owner asks the requester to produce a group signature on a nonce. The correct verification of the signature against the group public-key indicates the validity of the requester's role membership. The anonymity of the role member can be revoked with the help of the role administrator, who acts as a group manager in the group signature scheme.

To make authentication efficient in a distributed and ubiquitous computing environment, the group signature scheme has to be efficient. In early group signature schemes [13], group public keys grew with the size of the group and were inefficient. A group signature scheme with constant-sized public keys was first given in [12], and followed by a number of improvements [2, 3, 8, 9, 16, 22]. Recently, a group signature scheme was presented by Boneh, Boyen, and Shacham [8] that significantly shortens the signature length, compared to the RSA-based state-of-the-art group signature scheme by Ateniese *et al.* [2]. BBS group signature is under 200 bytes long and offers approximately the same level of security as a group signature five times longer in [2]. This improvement in the signature size significantly reduces the signature transmission time and storage space of mobile or small devices in a ubiquitous computing environment.

8 Related work

In this section, we compare our work with related decentralized trust management frameworks. Eschenauer, Gligor, and Baras presented a model for trust establishment in mobile *ad-hoc* environment [15]. An *ad-hoc* environment does not assume any pre-established role-management infrastructure such as role administrators and credentials. Therefore, the generation, issuance, and the distribution of trust evidence such as credentials are different from the context of the ubiquitous computing environment where pre-established role-management infrastructures are possible.

Trust evidences that are generated by recommendations and past experiences have been used for trust establishment in both *ad-hoc* and ubiquitous computing environments [6, 23, 25]. This type of trust evidences is flexible to collect, because it does not require any pre-established administration insfrastructure. However, it is difficult to use the abstraction of roles in these trust establishment model and thus the scalability of the model is low in comparison to the role-based trust management models such as the *RT* framework [18] and the RBCD model [24].

A number of trust management and distributed authorization systems have been proposed, for example KeyNote [7], delegation certificates [4], SPKI [14], Delegation Logic (DL) [17], proof-carrying authorization (PCA) [1], and *RT* framework [18]. Among them, *RT* framework is a *role-based* trust management framework for distributed environments. It supports decentralized authorization [18] and role-based delegation as in RBCD models. Our cRBCD model in Section 6 combines the credential chain discovery algorithms of the *RT* framework with the role-based cascaded delegation, which gives the original RBCD model more flexibility in constructing delegation chains.

The use of local (temporary) principal or key can be found in existing authorization systems such as DL [17] for controlling delegation scope. However, the main difference between our RBCD models and distributed authorization work [1, 17] is that RBCD allows an individual to delegate on behalf of a role and gives a mechanism to authenticate the delegator's role membership and thus the delegation validity. This features are not previously supported by any trust management or distributed authorization systems.

Although decentralized trust management systems, such as *RT*, are designed for coventional distributed computing environment, they can be adopted in ubiquitous computing environments for trust establishment across administrative domains. This is possible because these models allow the trust to be transferred through delegation chains. The delegation process in *RT* requires the participation of role administrators. This involves unnecessary communication overhead with the administrators. Therefore, it may not be efficient enough to accomodate the dynamic resource-sharing in the ubiquitous computing. In comparison, our RBCD model presented in this paper allows scalable, decentralized, and efficient role-based delegations without the participation of role administrators.

9 Conclusions

We have presented several credential and trust management techniques that improve the efficiency and flexibility of the role-based cascaded delegation model without increasing the management complexity. These improvements are important to the performance of delegation operations in a ubiquitous computing environment. The mRBCD model supports the merging of multiple delegation chains and improves the delegation efficiency. The cRBCD model improves the delegation flexibility by integrating the credential chain discovery with the cascaded delegation. We have also described how attribute constraints of a delegation chain are specified and verified, which are used in the cRBCD model to restrict the scope of the resource-sharing. Finally, we have discussed how the privacy protection of RBAC models in ubiquitous computing can be improved with the anonymous authentication of group identification techniques.

References

[1] A. W. Appel and E. W. Felten. Proof-carrying authentication. In *ACM Conference on Computer and Communications Security*, pages 52–62, 1999.

[2] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *Advances in Cryptology — CRYPTO '00*, volume 1880 of *LNCS*, pages 255–270. Springer Verlag, 2000.

[3] G. Ateniese, D. X. Song, and G. Tsudik. Quasi-efficient revocation in group signatures. In *Financial Cryptography '02*, volume 2357 of *LNCS*, pages 183–197.

[4] T. Aura. Distributed access-rights management with delegation certificates. In *Secure Internet Programming – Security Issues for Distributed and Mobile Objects*, volume 1603 of *LNCS*, pages 211–235. Springer, 1999.

[5] J. Bacon, K. Moody, and W. Yao. Access control and trust in the use of widely distributed services. In *Middleware 2001*, volume 2218 of *LNCS*, pages 295–310. Springer, 2001.

[6] B. Bhargava and Y. Zhong. Authorization based on evidence and trust. In *Proceedings of Data Warehouse and Knowledge Management Conference (DaWak-2002)*, September 2002.

[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] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Advances in Cryptology — Crypto '04*, LNCS, 2004.

[9] D. Boneh and M. Franklin. Anonymous authentication with subset queries (extended abstract). In *CCS '99: Proceedings of the 6th ACM conference on Computer and communications security*, pages 113–119. ACM Press, 1999.

[10] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology — Eurocrypt '03*, pages 416–432, 2003.

[11] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology — Asiacrypt '01*, volume 2248 of *LNCS*, pages 514–532. Springer-Verlag, 2001.

[12] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *CRYPTO '97*, volume 1296 of *LNCS*, pages 410–424. Springer-Verlag, 1997.

[13] D. Chaum and E. van Heijst. Group signatures. In *Advances in Cryptology — Eurocrypt '91*, pages 257–265. Springer-Verlag, 1991.

[14] 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.

[15] L. Eschenauer, V. D. Gligor, and J. Baras. On trust establishment in mobile ad-hoc networks. In *Proceedings of the Security Protocols Workshop*, April 2002.

[16] A. Kiayias and M. Yung. Extracting group signatures from traitor tracing schemes. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 630–648, Warsaw, Poland, 2003. Springer.

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

[18] 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.

[19] 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.

[20] K. Ren, T. Li, Z. Wan, F. Bao, R. H. Deng, and K. Kim. Highly reliable trust establishment scheme in ad hoc networks. *Computer Networks*, 45:687–699, 2004.

[21] 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.

[22] A. D. Santis, G. D. Crescenzo, and G. Persiano. Communication-efficient anonymous group identification. In *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security*, pages 73–82. ACM Press, 1998.

[23] B. Shand, N. Dimmock, and J. Bacon. Trust for ubiquitous, transparent collaboration. *Wirel. Netw.*, 10(6):711–721, 2004.

[24] R. Tamassia, D. Yao, and W. H. Winsborough. Role-based cascaded delegation. In *Proceedings of the ACM Symposium on Access Control Models and Technologies (SACMAT '04)*, pages 146 – 155. ACM Press, June 2004.

[25] G. Theodorakopoulos and J. S. Baras. Trust evaluation in ad-hoc networks. In *WiSe '04: Proceedings of the 2004 ACM workshop on Wireless security*, pages 1–10. ACM Press, 2004.