Distributed Credential Chain Discovery in Trust Management (Extended Abstract) [*]

Ninghui Li
Department of Computer Science, Gates 4B
Stanford University
Stanford, CA 94305-9045
ninghui.li@cs.stanford.edu

William H. Winsborough
NAI Labs
Network Associates, Inc.
3060 Washington Road
Glenwood, MD 21738
William_Winsborough@NAI.com

John C. Mitchell
Department of Computer Science, Gates 4B
Stanford University
Stanford, CA 94305-9045
mitchell@cs.stanford.edu

ABSTRACT

We give goal-oriented algorithms for discovering credential chains in RT_0, a role-based trust-management language introduced in this paper. The algorithms search credential graphs, a representation of RT_0 credentials. We prove that evaluation based on reachability in credential graphs is sound and complete with respect to the set-theoretic semantics of RT_0. RT_0 is more expressive than SDSI 2.0, so our algorithms can perform chain discovery in SDSI 2.0, for which existing algorithms in the literature either are not goal-oriented or require using specialized logic-programming inferencing engines. Being goal-oriented enables our algorithms to be used when credential storage is distributed. We introduce a type system for credential storage that guarantees well-typed, distributed credential chains can be discovered.

1. INTRODUCTION

Several trust-management systems have been proposed in recent years, *e.g.*, SPKI/SDSI [10], PolicyMaker [3, 4], KeyNote [2], Delegation Logic [15]. These systems are based on the notion of delegation, whereby one entity gives some of its authority to other entities. The process of making access control decisions involves finding a delegation chain from the source of authority to the requester. Thus, a central problem in trust management is to determine whether such a chain exists and, if so, to find it. We call this the *credential chain discovery problem*, by contrast with the *certificate chain discovery problem*, which is concerned with X.509 certificates [9]. Credentials in trust management generally have more complex meanings than simply binding names to public keys, and a credential chain is often a graph, rather than a linear path. The goal of this paper is to address the credential chain discovery problem (the *discovery problem* for short) in such systems.

Almost all existing work addressing the discovery problem assumes that potentially relevant credentials are all gathered in one place. This is at odds with the tenet of trust management—decentralized control; systems that use trust management typically issue and often store credentials in a distributed manner. This raises some nontrivial questions.

Example 1. *A fictitious Web publishing service, EPub, offers a discount to preferred customers of its parent organization, EOrg. EOrg issues a credential to the ACM stating that ACM members are preferred customers. Combining it with Alice's ACM membership credential yields a two-credential chain that proves Alice is a preferred customer. This is a linear chain; the subject of the credential issued by EOrg, ACM, is the issuer of the credential issued to Alice.*

These two credentials must be collected to construct the chain. The question we take up is where they should be stored to enable that collection. We say an entity A *stores* a credential if we can find the credential once we know A. Some other entity, such as a directory, may actually house the credential on A's behalf. Also, by storing a credential, we mean storing and providing access to the credential.

Given this definition of storing, to be useful, a credential must be stored with its issuer or with its subject. If both credentials in example 1 are stored with their subject, we can find them by obtaining the first credential from Alice, and using the issuer of that credential, ACM, to obtain the second. A disadvantage of this strategy is that it requires the ACM to store all the credentials authorizing ACM members. This makes the ACM a bottleneck. Also, some issuers may not entrust credentials to their subjects. If instead both credentials are stored with their issuers, the ACM has to store and provide all membership ids, again making it a bottle neck, and potentially causing broad search fan-out.

In the example, the ideal arrangement is to store one credential with EOrg and the other with Alice. The chain can then be discovered by working from these two ends towards the chain's middle. No prior credential discovery system supports this, probably because subject- and issuer-storage cannot be intermixed arbitrarily: in our example, if both credentials are stored exclusively by the ACM, the chain cannot be found. This is because in many decentralized systems, it is impossible or prohibitively expensive for one entity to enumerate all other entities in the systems. For all practical purposes, in such a system, if one can't find a

[*]This work is supported by DARPA through SPAWAR contracts N66001-00-C-8015 and N66001-01-C-8005, and AFRL/IF contract F30602-97-C-0336. A full version of this paper is available at: `http://crypto.stanford.edu/~ninghui/papers/disc.pdf`

credential chain without contacting every entity, one can't find it at all. In this paper, we introduce a credential typing system that constrains storage enough to ensure chains can be found by starting at their two ends and working inward.

The credential chain introduced in example 1 illustrates only the simplest case that we address. Some trust management systems, such as SDSI and Delegation Logic, allow what we call *attribute-based delegation*, that is the delegation of attribute authority to entities having certain attributes.

EXAMPLE 2. *EPub offers another discount to university students, and delegates the authority over the identification of students to entities that are accredited universities.*

Attribute-based delegation is achieved in SDSI through linked names, and in Delegation Logic through dynamic threshold structures and through conditional delegations. Systems that support attribute-based delegation promise high flexibility and scalability. However they significantly complicate the structure and discovery of credential chains.

Beyond storing credentials where they can be found, distributed discovery also requires an evaluation procedure that can drive credential collection. Such a procedure must be goal-oriented in the sense of expending effort only on chains that involve the requester and the access mediator, or its trusted authorities. In the Internet, with distributed storage of millions of credentials, most of them unrelated to one another, goal-oriented techniques will be crucial. The procedure must also be able to suspend evaluation, issue a request for credentials that could extend partial chains, and then resume evaluation when additional credentials are obtained. Existing evaluation procedures for SDSI and for Delegation Logic are either not goal-oriented, or do not support this alternation between collection and evaluation steps.

As a concrete foundation for discussing the discovery problem, we introduce a trust-management language, RT_0, which supports attribute-based delegation and subsumes SDSI 2.0 (the "SDSI" part of SPKI/SDSI 2.0 [10]). We provide goal-oriented evaluation algorithms based on a graphical representation of RT_0 credentials. This representation is ideal for driving credential collection because it makes it easy to suspend and resume, and to schedule work flexibly. Even in the centralized case, goal-orientation is an advantage when the credential pool is very large and contains many credentials that are unrelated. We also show how to use our algorithms to perform goal-oriented chain discovery for SDSI 2.0.

The rest of this paper is organized as follows. In section 2, we present the syntax and a set-theoretic semantics for RT_0. In section 3, we present goal-oriented, graph-based algorithms for centralized chain discovery in RT_0, and show how to apply them to SDSI as well. We prove that the graph-based notion of credential chains is sound and complete with respect to the semantics for RT_0. In section 4, we study chain discovery in the distributed case. We present a notion of well-typed credentials and prove that chains of well-typed credentials can always be discovered. In section 5, we discuss future directions and some related work. We conclude in section 6.

2. A ROLE-BASED TRUST-MANAGEMENT LANGUAGE

This section introduces RT_0, the first (and the simplest) in a series of role-based trust-management languages we are developing. We present RT_0's syntax, discuss its intended meaning, and compare it to SDSI. Then we give a formal semantics.

2.1 The Language RT_0

The constructs of RT_0 include entities, role names, and roles. Typically, an *entity* is a public key, but could also be, say, a user account. Entities can issue credentials and make requests. RT_0 requires that each entity can be uniquely identified and that one can determine which entity issued a particular credential or a request. In this paper, we use A, B, and D to denote entities. A *role name* is an identifier, say, a string. We use r, r_1, r_2, *etc.*, to denote role names. A *role* takes the form of an entity followed by a role name, separated by a dot, *e.g.*, $A.r$ and $B.r_1$. The notion of roles is central in RT_0. A role has a value that is a set of entities who are members of this role. Each entity A has the authority to define who are the members of each role of the form $A.r$. A role can also be viewed as an attribute. An entity is a member of a role if and only if it has the attribute identified by the role. In RT_0, an access control permission is represented as a role as well. For example, the permission to shut down a computer can be represented by a role OS.shutdown.

There are four kinds of credentials in RT_0, each corresponding to a different way of defining role membership:

- *Type-1*: $A.r \longleftarrow B$

  A and B are (possibly the same) entities, and r is a role name.

  This means that A defines B to be a member of A's r role. In the attribute-based view, this credential can be read as B has the attribute $A.r$, or equivalently, A says that B has the attribute r.

- *Type-2*: $A.r \longleftarrow B.r_1$

  A and B are (possibly the same) entities, and r and r_1 are (possibly the same) role names.

  This means that A defines its r role to include all members of B's r_1 role. In other words, A defines the role $B.r_1$ to be more powerful than $A.r$, in the sense that a member of $B.r_1$ can do anything that the role $A.r$ is authorized to do. Such credentials can be used to define role-hierarchy in Role-Based Access Control (RBAC) [16]. The attribute-based reading of this credential is: if B says that an entity has the attribute r_1, then A says that it has the attribute r. In particular, if r and r_1 are the same, this is a delegation from A to B of authority over r.

- *Type-3*: $A.r \longleftarrow A.r_1.r_2$

  A is an entity, and r, r_1, and r_2 are role names. We call $A.r_1.r_2$ a *linked role*.

  This means that $\mathit{members}(A.r) \supseteq \mathit{members}(A.r_1.r_2) = \bigcup_{B \in \mathit{members}(A.r_1)} \mathit{members}(B.r_2)$, where $\mathit{members}(e)$ represents the set of entities that are members of e. The attribute-based reading of this credential is: if A says that an entity B has the attribute r_1, and B says that an entity D has the attribute r_2, then A says that D has the attribute r. This is attribute-based delegation: A identifies B as an authority on r_2 not by using (or knowing) B's identity, but by another attribute of B (*viz.*, r_1). If r and r_2 are the same, A is delegating its authority over r to anyone that A believes to have the attribute r_1.

- *Type-4*: $A.r \longleftarrow f_1 \cap f_2 \cap \cdots \cap f_k$

  A is an entity, k is an integer greater than 1, and each f_j, $1 \leq j \leq k$, is an entity, a role, or a linked role starting with A. We call $f_1 \cap f_2 \cap \cdots \cap f_k$ an *intersection*.

  This means that $members(A.r) \supseteq (members(f_1) \cap \cdots \cap members(f_k))$. The attribute-based reading of this credential is: anyone who has all the attributes $f_1, \ldots, f_k$ also has the attribute $A.r$.

A *role expression* is an entity, a role, a linked role, or an intersection. We use e, e_1, e_2, *etc*, to denote role expressions. By contrast, we use $f_1, \ldots, f_k$ to denote the intersection-free expressions occurring in intersections. All credentials in RT_0 take the form, $A.r \longleftarrow e$, where e is a role expression. Such a credential means that $members(A.r) \supseteq members(e)$, as we formalize in section 2.2 below. We say that this credential *defines* the role $A.r$. (This choice of terminology is motivated by analogy to name definitions in SDSI, as well as to predicate definitions in logic programming.) We call A the *issuer*, e the *right-hand side*, and each entity in $base(e)$ a *subject* of this credential, where $base(e)$ is defined as follows: $base(A) = \{A\}$, $base(A.r) = \{A\}$, $base(A.r_1.r_2) = \{A\}$, $base(f_1 \cap \cdots \cap f_k) = base(f_1) \cup \cdots \cup base(f_k)$.

Example 3. *Combining examples 1 and 2, EPub offers a special discount to anyone who is both a preferred customer of EOrg and a student. To identify legitimate universities, EPub accepts accrediting credentials issued by the fictitious Accrediting Board for Universities (ABU). The following credentials prove Alice is eligible for the special discount:*

$$\left\{\begin{array}{c} \text{EPub.spdiscount} \longleftarrow \text{EOrg.preferred} \cap \text{EPub.student}, \\ \text{EOrg.preferred} \longleftarrow \text{ACM.member}, \\ \text{ACM.member} \longleftarrow \text{Alice}, \\ \text{EPub.student} \longleftarrow \text{EPub.university.stuID}, \\ \text{EPub.university} \longleftarrow \text{ABU.accredited}, \\ \text{ABU.accredited} \longleftarrow \text{StateU}, \\ \text{StateU.stuID} \longleftarrow \text{Alice} \end{array}\right\}$$

Readers familiar with Simple Distributed Security Infrastructure (SDSI) [8, 10] may notice the similarity between RT_0 and SDSI's name certificates. Indeed, our design is heavily influenced by existing trust-management systems, especially SDSI and Delegation Logic (DL) [15]. RT_0 can be viewed as an extension to SDSI 2.0 or a syntactically sugared version of a subset of DL. The arrows in RT_0 credentials are the reverse direction of those in SPKI/SDSI. We choose to use this direction to be consistent with an underlying logic programming reading of credentials and with directed edges in credential graphs, introduced below in section 3. In addition, RT_0 differs from SDSI 2.0 in the following two aspects.

First, SDSI allows arbitrarily long linked names, while we allow only length-2 linked roles. There are a couple of reasons for this design. We are not losing any expressive power; one can always break up a long chain by introducing additional roles and credentials. Moreover, it often makes sense to break long chains up, as doing so creates more modular policies. If A wants to use $B.r_1.r_2.\cdots.r_k$ in its credential, then $B.r_1.r_2.\cdots.r_{k-1}$ must mean something to A; otherwise, why would A delegate power to members of $B.r_1.r_2.\cdots.r_{k-1}$? Having to create a new role makes A think about what $B.r_1.r_2.\cdots.r_{k-1}$ means. Finally, restricting lengths of linked roles simplifies the design of algorithms for chain discovery.

Second, SDSI doesn't have RT_0's type-4 credentials, and so RT_0 is more expressive than the current version of SDSI 2.0. Intersections and threshold structures (*e.g.*, those in [10]) can be used to implement one another. Threshold structures may appear in name certificates according to [10] and earlier versions of [11]. This is disallowed in [8] and the most up-to-date version of [11], because threshold structures are viewed as too complex [8]. Intersections provide similar functionality with simple and clear semantics.

2.2 The Semantics of RT_0

This section presents a non-operational semantics of RT_0. Given a set $\mathcal{C}$ of RT_0 credentials, we define a map $\mathcal{S}_\mathcal{C}$: $\mathsf{Roles} \rightarrow \wp(\mathsf{Entities})$, where $\wp(\mathsf{Entities})$ is the power set of $\mathsf{Entities}$. $\mathcal{S}_\mathcal{C}$ is given by the least solution to a system of set inequalities that is parameterized by a finite, set-valued function, $\mathsf{rmem} : \mathsf{Roles} \rightarrow \wp(\mathsf{Entities})$. That is, the semantics is the least such function that satisfies the system, where the ordering is pointwise subset. We use a least fixpoint so as to resolve circular role dependencies. To help construct the system of inequalities, we extend rmem to arbitrary role expressions (whose domain we denote by $\mathsf{RoleExpressions}$) through the use of an auxiliary semantic function, $\mathsf{expr}_{\mathsf{rmem}}$: $\mathsf{RoleExpressions} \rightarrow \wp(\mathsf{Entities})$ defined as follows:

$$\begin{array}{rcl} \mathsf{expr}_{\mathsf{rmem}}(B) & = & \{B\} \\ \mathsf{expr}_{\mathsf{rmem}}(A.r) & = & \mathsf{rmem}(A.r) \\ \mathsf{expr}_{\mathsf{rmem}}(A.r_1.r_2) & = & \bigcup_{B \in \mathsf{rmem}(A.r_1)} \mathsf{rmem}(B.r_2) \\ \mathsf{expr}_{\mathsf{rmem}}(f_1 \cap \cdots \cap f_k) & = & \bigcap_{1 \leq j \leq k} \mathsf{expr}_{\mathsf{rmem}}(f_j) \end{array}$$

We now define $\mathcal{S}_\mathcal{C}$ to be the least value of rmem satisfying the following system of inequalities:

$$\left\{ \ \mathsf{expr}_{\mathsf{rmem}}(e) \subseteq \mathsf{rmem}(A.r) \;\middle|\; A.r \longleftarrow e \in \mathcal{C} \ \right\}.$$

As with rmem, we use expr to extend $\mathcal{S}_\mathcal{C}$ to role expressions, writing $\mathsf{expr}_{\mathcal{S}_\mathcal{C}}(e)$ for the members of role expression e.

The least solution to such a system can be constructed as the limit of a sequence $\{\mathsf{rmem}_i\}_{i \in \mathcal{N}}$, where $\mathcal{N}$ is the set of natural numbers, and where for each i, $\mathsf{rmem}_i : \mathsf{Roles} \rightarrow \wp(\mathsf{Entities})$. The sequence is defined inductively by taking $\mathsf{rmem}_0(A.r) = \emptyset$ for each role $A.r$ and by defining rmem_{i+1} so that for each role $A.r$,

$$\mathsf{rmem}_{i+1}(A.r) = \bigcup_{A.r \longleftarrow e \,\in\, \mathcal{C}} \mathsf{expr}_{\mathsf{rmem}_i}(e).$$

The function that relates the values of $\{\mathsf{rmem}_i\}_{i \in \mathcal{N}}$ is monotonic, because the operators used to construct it ($\cap$ and $\cup$) are monotonic. Furthermore, $\mathsf{Roles} \rightarrow \wp(\mathsf{Entities})$ is a complete lattice. So this sequence is known to converge to the function's least fixpoint, which is clearly also the least solution to the inequalities. (As the lattice is finite, convergence takes place finitely.) Thus, the least solution exists and is easily constructed. For instance, referring to example 3 and showing only changes in the function's value, successive values of rmem_i have: for $i = 1$, ABU.accredited = {StateU}, StateU.stuID = {Alice}, ACM.member = {Alice}; for $i = 2$, EPub.university = {StateU}, EOrg.preferred = {Alice}; for $i = 3$, EPub.student = {Alice}; for $i = 4$, EPub.spdiscount = {Alice}, where they stabilize.

3. CENTRALIZED CHAIN DISCOVERY

Given a set of credentials $\mathcal{C}$ in RT_0, three important kinds of queries are:

1. Given a role $A.r$, determine its member set, $\mathcal{S}_{\mathcal{C}}$(A.r);
2. Given an entity D, determine all the roles it belongs to, *i.e.*, all role $A.r$'s such that $D \in \mathcal{S}_{\mathcal{C}}(A.r)$;
3. Given a role $A.r$ and an entity D, determine whether $D \in \mathcal{S}_{\mathcal{C}}(A.r)$.

In this section, we study credential chain discovery for RT_0 when credentials are centralized. We give goal-oriented algorithms for answering the above three kinds of queries.

3.1 Algorithm Requirements

Chain discovery in RT_0 shares two key problem characteristics with discovery in SDSI: linked names give credential chains a non-linear structure and role definitions can be cyclic. Cyclic dependencies must be managed to avoid non-termination. Clarke *et al.* [8] have given an algorithm for chain discovery in SPKI/SDSI 2.0. Their algorithm views each certificate as a rewriting rule and views discovery as a term-rewriting problem. It manages cyclic dependency by using a bottom-up approach—it performs a closure operation over the set of all credentials before it finds one chain. This may be suitable when large numbers of queries are made about a slowly changing credential pool of modest size. However, as the frequency of changes to the credential pool (particularly deletions, such as credential expirations or revocations) approaches the frequency of queries against the pool, the efficiency of the bottom-up approach deteriorates rapidly, particularly when pool size is large.

Li [14] gave a 4-rule logic program to calculate meanings of SDSI credentials. Cyclic dependencies are managed by using XSB [17] to evaluate the program. XSB's extension table mechanism avoids non-termination problems to which other Prolog engines succumb. Yet, for many trust-management applications, this solution is excessively heavyweight. Moreover, in its current form, the resulting evaluation mechanism cannot be used to drive credential collection.

As discussed in section 1, because we seek techniques that work well when the credential pool is distributed or changes frequently, we require chain discovery algorithms that are goal-directed and that can drive the collection process. They also must support interleaving credential collection and chain construction (*i.e.*, evaluation) steps.

We meet these requirements by providing graph-based evaluation algorithms. Credentials are represented by edges. Chain discovery is performed by starting at the node representing the requester, or the node representing the role (permission) to be proven, or both, and then traversing paths in the graph trying to build an appropriate chain. In addition to being goal-directed, this approach allows the elaboration of the graph to be scheduled flexibly. Also, the graphical representation of the evaluation state makes it relatively straightforward to manage cyclic dependencies. To our knowledge, our algorithms are the first to use a graphical representation to handle linked roles.

3.2 A Graph Representation of Credentials

We define a directed graph, which we call a *credential graph*, to represent a set of credentials and their meanings. Each node in the graph represents a role expression occurring in a credential in $\mathcal{C}$. Every credential $A.r \longleftarrow e \in \mathcal{C}$ contributes an edge $e \rightarrow A.r$.[1] (This holds for credentials of all types.) The destinations of these edges are roles. Edges are also added whose destinations are linked roles and intersections. We call these *derived* edges because their inclusion come from the existence of other, semantically related, paths in the graph.

Definition 1 (Credential Graph). *For a set of credentials $\mathcal{C}$, the corresponding credential graph is given by $G_{\mathcal{C}} = \langle N_{\mathcal{C}}, E_{\mathcal{C}} \rangle$ where $N_{\mathcal{C}}$ and $E_{\mathcal{C}}$ are defined as follows.*

$$N_{\mathcal{C}} = \bigcup_{A.r \longleftarrow e \,\in\, \mathcal{C}} \{A.r, e\}.$$

$E_{\mathcal{C}}$ is the least set of edges over $N_{\mathcal{C}}$ that satisfies the following three closure properties:

Closure Property 1: *If $A.r \longleftarrow e \in \mathcal{C}$, then $e \rightarrow A.r \in E_{\mathcal{C}}$.*

Closure Property 2: *If $B.r_2$, $A.r_1.r_2 \in N_{\mathcal{C}}$ and there is a path $B \xrightarrow{*} A.r_1$ in $E_{\mathcal{C}}$, then $B.r_2 \rightarrow A.r_1.r_2 \in E_{\mathcal{C}}$; we say that this edge* is derived from *the path $B \xrightarrow{*} A.r_1$.*

Closure Property 3: *If D, $f_1 \cap \cdots \cap f_k \in N_{\mathcal{C}}$ and for each $j \in [1..k]$ there is a path $D \xrightarrow{*} f_j$, then $D \rightarrow f_1 \cap \cdots \cap f_k \in E_{\mathcal{C}}$; we say that this edge is derived from the paths $D \xrightarrow{*} f_j$, for $j \in [1..k]$.*

This definition can be made effective by inductively constructing a sequence of edge sets $\{{E_{\mathcal{C}}}^i\}_{i \in \mathcal{N}}$ whose limit is $E_{\mathcal{C}}$. We take ${E_{\mathcal{C}}}^0 = \{e \rightarrow A.r \mid A.r \longleftarrow e \in \mathcal{C}\}$ and construct ${E_{\mathcal{C}}}^{i+1}$ from ${E_{\mathcal{C}}}^i$ by adding one edge according to either closure property 2 or 3. Since $\mathcal{C}$ is finite, we do not have to worry about scheduling these additions. At some finite stage, no more edges will be added, and the sequence converges to $E_{\mathcal{C}}$.

Theorem 1 (Soundness). *Given an entity D and a role expression e, if there is a path $D \xrightarrow{*} e$ in $E_{\mathcal{C}}$, then $D \in \mathsf{expr}_{\mathcal{S}_{\mathcal{C}}}(e)$.*

Proof. The proof is by induction on the steps of the construction of $\{{E_{\mathcal{C}}}^i\}_{i \in \mathcal{N}}$ shown above. We prove an induction hypothesis that is slightly stronger than the theorem: For each $i \in \mathcal{N}$ and for any role expressions e_1 and e, if there is a path $e_1 \xrightarrow{*} e$ in ${E_{\mathcal{C}}}^i$, then $\mathsf{expr}_{\mathcal{S}_{\mathcal{C}}}(e_1) \subseteq \mathsf{expr}_{\mathcal{S}_{\mathcal{C}}}(e)$.

We show the base case by using a second, inner induction on the length of the path $e_1 \xrightarrow{*} e$ in ${E_{\mathcal{C}}}^0$. The inner base case, in which $e_1 = e$, is trivial; we consider the step. Suppose $(e_1 \xrightarrow{*} e) = (e_1 \xrightarrow{*} e_2 \rightarrow e)$. Because each edge in ${E_{\mathcal{C}}}^0$ corresponds to a credential, we have $e \longleftarrow e_2 \in \mathcal{C}$. It follows that $\mathsf{expr}_{\mathcal{S}_{\mathcal{C}}}(e_2) \subseteq \mathsf{expr}_{\mathcal{S}_{\mathcal{C}}}(e)$, by definition of $\mathcal{S}_{\mathcal{C}}$. The induction assumption gives us $\mathsf{expr}_{\mathcal{S}_{\mathcal{C}}}(e_1) \subseteq \mathsf{expr}_{\mathcal{S}_{\mathcal{C}}}(e_2)$, so $\mathsf{expr}_{\mathcal{S}_{\mathcal{C}}}(e_1) \subseteq \mathsf{expr}_{\mathcal{S}_{\mathcal{C}}}(e)$.

We prove the step by again using an inner induction on the length of $e_1 \xrightarrow{*} e$, which we now assume is in ${E_{\mathcal{C}}}^{i+1}$. Again the basis is trivial. For the step, we decompose $e_1 \xrightarrow{*} e$ into $e_1 \xrightarrow{*} e_2 \rightarrow e$. There are three cases, depending on which closure property introduced the edge $e_2 \rightarrow e$.

case 1: When $e_2 \rightarrow e$ is introduced by closure property 1, the argument proceeds along the same lines as the base case, using the inner induction hypothesis on $e_1 \xrightarrow{*} e_2$ to derive $\mathsf{expr}_{\mathcal{S}_{\mathcal{C}}}(e_1) \subseteq \mathsf{expr}_{\mathcal{S}_{\mathcal{C}}}(e_2)$.

[1]While long, lefthand arrows ($\longleftarrow$) represent credentials, short, righthand arrows ($\rightarrow$) represent edges, and short, righthand arrows with stars ($\xrightarrow{*}$) represent paths, which consist of zero or more edges.

case 2: When $e_2 \rightarrow e$ is introduced by closure property 2, e has the form $A.r_1.r_2$, e_2 has the form $B.r_2$, and there is a path $B \xrightarrow{*} A.r_1$ in ${E_\mathcal{C}}^i$. The outer induction hypothesis gives us $\mathsf{expr}_{\mathcal{S}_\mathcal{C}}(B) \subseteq \mathsf{expr}_{\mathcal{S}_\mathcal{C}}(A.r_1)$, *i.e.*, $B \in \mathcal{S}_\mathcal{C}(A.r_1)$. The inner induction hypothesis gives us $\mathsf{expr}_{\mathcal{S}_\mathcal{C}}(e_1) \subseteq \mathsf{expr}_{\mathcal{S}_\mathcal{C}}(B.r_2)$. Together with the definition of expr for $A.r_1.r_2$, these imply $\mathsf{expr}_{\mathcal{S}_\mathcal{C}}(e_1) \subseteq \mathsf{expr}_{\mathcal{S}_\mathcal{C}}(e)$, as required.
case 3: When $e_2 \rightarrow e$ is introduced by closure property 3, e has the form $f_1 \cap \cdots \cap f_k$, $e_2 = e_1$ is an entity D (because entity nodes have no incoming edges), and there are paths $D \xrightarrow{*} f_j$ in ${E_\mathcal{C}}^i$ for each $j \in [1..k]$. The outer induction hypothesis gives us $D \in \mathsf{expr}_{\mathcal{S}_\mathcal{C}}(f_j)$ for $j \in [1..k]$; therefore, $\mathsf{expr}_{\mathcal{S}_\mathcal{C}}(e_1) \subseteq \mathsf{expr}_{\mathcal{S}_\mathcal{C}}(e)$. □

THEOREM 2 (COMPLETNESS). *For any role, $A.r$, $D \in \mathcal{S}_\mathcal{C}(A.r)$ implies there exists a path $D \xrightarrow{*} A.r$ in $E_\mathcal{C}$.*

The proofs for this and other theorems are omitted due to space limitation; they can be found in the full version of this paper.

Together, Theorems 1 and 2 tell us that we can answer each of the queries enumerated at the top of this section by consulting the credential graph. The rest of this section gives algorithms for constructing subgraphs that enable us to answer such questions without constructing the entire graph. As we have seen, constructing the path $D \xrightarrow{*} A.r$ alone proves D is in role $A.r$. However, where $D \xrightarrow{*} A.r$ contains derived edges, the paths they are derived from must be constructed first. The portion of the credential graph that must be constructed is what we call a *credential chain*: *chain*$(D \xrightarrow{*} A.r)$ is the least set of edges in $E_\mathcal{C}$ containing $D \xrightarrow{*} A.r$ and also containing all the paths that the derived edges in the set are derived from.

3.3 The Backward Search Algorithm

The backward search algorithm determines the member set of a given role expression e_0. In terms of the credential graph, it finds all the entity nodes that can reach the node e_0. We call it backward because it follows edges in the reverse direction. This name is consistent with the terminology in X.509 [5, 9], in which forward means going from subjects to issuers and reverse means from issuers to subjects. This algorithm works by constructing *proof graphs*, which are equivalent to, but slightly different from, subgraphs of a credential graph. The minor difference is discussed after the presentation of the algorithm.

The backward search algorithm constructs a proof graph, maintaining a queue of nodes to be processed; both initially contain just one node, e_0. Nodes are processed one by one until the queue is empty.

To process a role node $A.r$, the algorithm finds all credentials that define $A.r$. For each credential $A.r \longleftarrow e$, it creates a node for e, if none exists, and adds the edge $e \rightarrow A.r$. In the proof graph, there is only one node corresponding to each role expression and each edge is added only once. Each time the algorithm tries to create a node for a role expression e, it first checks whether such a node already exists; if not, it creates a new node, adds it into the queue, and returns it. Otherwise, it returns the existing node.

On each node e, the algorithm stores a children set, which is a set of nodes, e_1, that e can reach directly (*i.e.*, $e \rightarrow e_1$), and a solution set, which is the set of entity nodes, D, that can reach e (*i.e.*, $D \xrightarrow{*} e$). Solutions are propagated from e to e's children in the following ways. When a node is notified to add a solution, it checks whether the solution exists in its solution set; if not, it adds the solution and then notifies all its children about this new solution. When a node e_1 is first added as a child of e_2 (as the result of adding $e_2 \rightarrow e_1$), all existing solutions on e_2 are copied to e_1.

To process an entity node, the algorithm notifies the node to add itself to its own solution set.

To process a linked role node $A.r_1.r_2$, the algorithm creates a node for $A.r_1$ and creates a *linking monitor* to observe the node. The monitor, on observing that $A.r_1$ has received a new solution B, creates a node for $B.r_2$ and adds the edge $B.r_2 \rightarrow A.r_1.r_2$, which we call a link-containment edge.

To process an intersection node $e = f_1 \cap \cdots \cap f_k$, the algorithm creates one *intersection monitor*, for e, and k nodes, one for each f_j, then makes the monitor observe each node f_j. This monitor counts how many times it observes that an entity D is added. For a given entity D, each f_j notifies e at most once. If the count reaches k, then the monitor adds the edge $D \rightarrow e$. So, to summarize, in addition to the nodes and edges in the credential graph, the algorithm constructs monitors that implement closure properties 2 and 3.

Given a set of credentials $\mathcal{C}$, the proof graph, $G_{b(e_0,\mathcal{C})}$, constructed by the backward search algorithm starting from e_0, is closely related to the credential graph, $G_\mathcal{C}$. $G_{b(e_0,\mathcal{C})}$ is almost identical to the smallest subgraph of $G_\mathcal{C}$ whose node set, $N^0_\mathcal{C}$, satisfies the following four closure properties and whose edge set consists of all edges of $E_\mathcal{C}$ over nodes of $N^0_\mathcal{C}$: (*i*) $e_0 \in N^0_\mathcal{C}$; (*ii*) $e_2 \in N^0_\mathcal{C}$ & $e_1 \rightarrow e_2 \in E_\mathcal{C} \implies e_1 \in N^0_\mathcal{C}$; (*iii*) $A.r_1.r_2 \in N^0_\mathcal{C} \implies A.r_1 \in N^0_\mathcal{C}$; and (*iv*) $f_1 \cap \cdots \cap f_k \in N^0_\mathcal{C}$ & $j \in [1..k] \implies f_j \in N^0_\mathcal{C}$. The only difference between $G_{b(e_0,\mathcal{C})}$ and such a subgraph of $G_\mathcal{C}$ is this: $G_{b(e_0,\mathcal{C})}$ contains role nodes, created during the processing of linked roles, that don't appear in $\mathcal{C}$. Specifically, when the algorithm processes a linked-role node $A.r_1.r_2$, the node $B.r_2$ and the link-containment edge, $B.r_2 \rightarrow A.r_1.r_2$, are added, even when $B.r_2$ does not appear in $\mathcal{C}$, and will therefore receive no incoming edges and no solutions. It is not difficult to see that $G_{b(e_0,\mathcal{C})}$ contains *chain*$(D \xrightarrow{*} e_0)$ for every D that can reach e_0.

THEOREM 3. *Given a set of credentials $\mathcal{C}$, let N be the number of credentials in $\mathcal{C}$, and M be the total size of $\mathcal{C}$: $\sum_{A.r \longleftarrow e \in \mathcal{C}} |e|$, where $|A| = |A.r| = |A.r_1.r_2| = 1$ and $|f_1 \cap \cdots \cap f_k| = k$. Assuming that finding all credentials that define a role takes time linear in the number of such credentials (e.g., by using hashing), then the worst-case time complexity of the backward search algorithm is $O(N^3 + NM)$, and the space complexity is $O(NM)$. If each intersection in $\mathcal{C}$ has size $O(N)$, then the time complexity is $O(N^3)$.*

To see that $O(N^3)$ is a tight bound for the algorithm, consider the following example:
$\mathcal{C} = \{A_0.r_0 \longleftarrow A_i,\ A_0.r_i \longleftarrow A_0.r_{i-1 \bmod n},\ A_i.r_0 \longleftarrow A_{i-1 \bmod n}.r_0,\ A_0.r' \longleftarrow A_0.r_i.r_0 \mid 0 \leq i < n\}$
There are $N = 4n$ credentials. When using backward search algorithm from $A_0.r'$, there are edges from each $A_j.r_0$ to each $A_0.r_i.r_0$, where $0 \leq i, j < n$, so there are n^2 such edges. Each $A_j.r_0$ gets n solutions, so the time complexity is n^3. We can see that intersections do not increase the worst-case time complexity of this algorithm. $O(NM)$ is a tight space bound. Following is an example that reaches the bound: $\mathcal{C} = \{A_0.r_0 \longleftarrow A_i,\ A_0.r_i \longleftarrow A_0.r_{i-1 \bmod n},\ A_0.r' \longleftarrow A_0.r_i.r_0 \cap A_0.r_i.r_1 \cap \cdots \cap A_0.r_i.r_{K-1} \mid 0 \leq i < n\}$

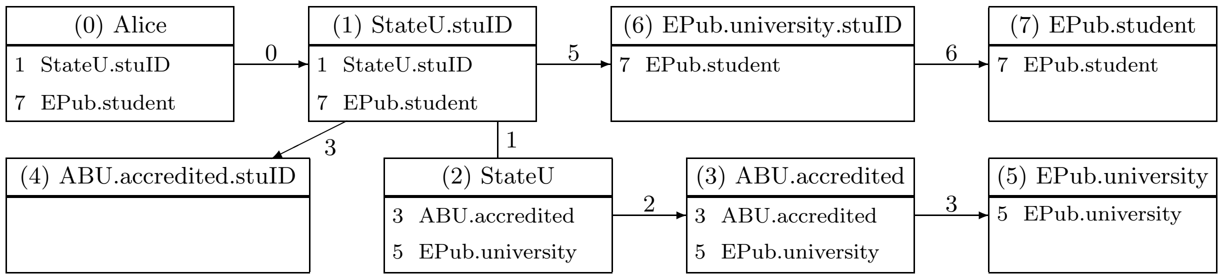

Figure 1: $G_{f(\textbf{Alice},\mathcal{C})}$, the proof graph constructed by doing forward search from Alice with $\mathcal{C} = \{\textbf{EPub.student} \longleftarrow \textbf{EPub.university.student}, \textbf{EPub.university} \longleftarrow \textbf{ABU.accredited}, \textbf{ABU.accredited} \longleftarrow \textbf{StateU}, \textbf{StateU.stuID} \longleftarrow \textbf{Alice}\}$. The first line of each node gives the node number in order of creation and the role expression represented by the node. The second part of a node lists each solution eventually associated with this node. Each of those solutions and each graph edge is labeled by the number of the node that was being processed when the solution or edge was added. The edge labeled with 1 is a linking monitor.

3.4 The Forward Search Algorithm

The forward search algorithm finds all roles that an entity is a member of. The direction of the search moves from the subject of a credential towards its issuer.

The forward algorithm has the same overall structure as the backward algorithm; however, there are some differences. First, each node stores its parents instead of its children. Second, each node e stores two kinds of solutions: full solutions and partial solutions. Each *full solution* on e is a role that e is a member of, *i.e.*, a role node that is reachable from e. Each *partial solution* has the form $(f_1 \cap \cdots \cap f_k,\, j)$, where $1 \leq j \leq k$. The node e gets the solution $(f_1 \cap \cdots \cap f_k, j)$ when f_j is reachable from e. Such a partial solution is just one piece of a proof that e can reach $f_1 \cap \cdots \cap f_k$. It is passed through edges in the same way as is a full solution. When an entity node D gets the partial solution, it checks whether it has all k pieces; if it does, it creates a node for $f_1 \cap \cdots \cap f_k$, if none exists, and adds the edge $D \rightarrow f_1 \cap \cdots \cap f_k$.

The processing of each node is also different from that in the backward algorithm. For any role expression e, forward processing involves the following three steps. First, if e is a role $B.r_2$, add itself as a solution to itself, then add a linking monitor observing B. This monitor, when B gets a full solution $A.r_1$, creates the node $A.r_1.r_2$ and adds the edge $B.r_2 \rightarrow A.r_1.r_2$. The addition of such an edge results in $B.r_2$ being added as a parent of $A.r_1.r_2$. Second, find all credentials of the form $A.r \longleftarrow e$; for each such credential, create a node for $A.r$, if none exists, and add the edge $e \rightarrow A.r$. Third, if e is not an intersection, find all credentials of the form $A.r \longleftarrow f_1 \cap \cdots \cap f_k$ such that some $f_j = e$; then add $(f_1 \cap \cdots \cap f_k,\, j)$ as a partial solution on e.

Figure 1 shows the result of doing forward search using a subset of the credentials in example 3.

THEOREM 4. *Under the same assumptions as in theorem 3, the time complexity for the forward search algorithm is $O(N^2M)$, and the space complexity is $O(NM)$.*

3.5 Bi-direction Search Algorithms

When answering queries about whether a given entity, D, is a member of a given role, $A.r$, we have the flexibility of combining forward and backward algorithms into a search that proceeds from both D and $A.r$ at once. In this bi-directional algorithm, a node e stores both its parents and its children, as well as both backward solutions (entities that are members of e) and forward solutions (roles that e is a member of).

In the centralized case, doing either forward search from D or backward search from $A.r$ suffices to answer the query. However, using bi-directional search could improve search efficiency (where search space size is sometimes exponential in path length) by finding two shorter intersecting paths, rather than one longer one. A variety of search strategies bear consideration, and different algorithms can be developed based on them. The algorithms described above use queues to organize node processing, resulting in breadth-first search. If they used stacks, they would perform depth-first search. In general, when there are several nodes that can be explored (from either direction), they can be placed in a priority queue according to some heuristic criteria, *e.g.*, fan-out. Note that these remarks also apply to the forward and backward algorithms.

In the distributed case, the ability to locate credentials can become a limiting factor. This is the main issue we address in section 4.

3.6 Implementation, Generalization, and Application to SDSI

We have implemented the above algorithms in Java. Our program can be configured to store the parent or child node from which each solution arrives. Using this information, one can easily trace paths, and compute the set of credentials being used in any proof graph.

Our algorithms can be generalized to search for paths between two arbitrary role expressions. One way to do this is to generalize the solution set to collect all reachable nodes, not just entity and role nodes. Then, one knows that a path $e_1 \overset{*}{\rightarrow} e_2$ exists when e_1 is added as a backward solution on e_2 or when e_2 is added as a forward solution on e_1. Of course, such a change would affect the algorithm's complexity.

Our algorithms can also be used to do chain discovery in SDSI. To allow their construction in RT_0, long linked names can be broken up. Instead of using $A.r \longleftarrow B.r_1.r_2.\cdots.r_k$, one can use $\{A.r \longleftarrow A.r'_{k-1}.r_k,\, A.r'_{k-1} \longleftarrow A.r'_{k-2}.r_{k-1}, \cdots,\, A.r'_2 \longleftarrow A.r'_1.r_2,\, A.r'_1 \longleftarrow B.r_1\}$, in which the r'_i's are newly introduced role names. Then one can use any of the algorithms to do goal-oriented chain discovery.

THEOREM 5. *Given a set of "SDSI" credentials $\mathcal{C}$, which have arbitrarily long linked roles and no intersection, let*

$\mathcal{C}'$ be the result of breaking up long linked roles. Then the time complexity of the backward algorithm, applied to $\mathcal{C}'$, is $O(N^3L)$, where N is the number of credentials in $\mathcal{C}$, and L is the length of the longest linked role in $\mathcal{C}$.

This $O(N^3L)$ worst-case complexity is the same as that of the algorithm in Clarke *et al.* [8].

Instead of breaking up long linked names, one can extend our algorithms to handle them directly. It is also not difficult to extend our algorithms to handle SPKI delegation certificates. In particular, it is straightforward to extend our techniques for handling intersections to handle threshold structures as well.

4. DISTRIBUTED CHAIN DISCOVERY

The algorithms given in the previous section can be used when credential storage is not centralized, but distributed among credentials' subjects and issuers. As discussed in section 1, it is impractical to require either that all credentials be stored by their issuers or that all be stored by their subjects. Yet if no constraint is imposed on where credentials are stored, some chains cannot be found without broadcast, which we assume is unavailable.

EXAMPLE 4. *Consider the following credentials from example 3: $ABU.accredited \longleftarrow StateU$ and $StateU.stuID \longleftarrow Alice$. If both of these are stored exclusively with $StateU$, none of our search procedures can find the chain that authorizes Alice. Arriving at ABU and at $Alice$, the procedure is unable to locate either of these two key credentials.*

This section presents a type system for credential storage that ensures chains of well-typed credentials can be found.

4.1 Traversability

We introduce notions of path traversability to formalize the three different directions in which distributed chains can be located and assembled, depending on the storage characteristics of their constituent credentials. We call the three notions, *forward traversability*, *backward traversability*, and *confluence*, respectively. Working from one end or the other, or from both simultaneously, a search agent needs to be able to find the credential defining each edge in a path, $D \overset{*}{\to} A.r$, as well as in the other paths of $chain(D \overset{*}{\to} A.r)$, which prove the existence of derived edges in $D \overset{*}{\to} A.r$.

Suppose that $D \overset{*}{\to} A.r$ consists entirely of edges that represent credentials that are stored by their subjects. (In this case, $(D \overset{*}{\to} A.r) = chain(D \overset{*}{\to} A.r)$.) We call $D \overset{*}{\to} A.r$ *forward traversable* because forward search can drive its distributed discovery, as follows. Obtain from D the first credential of the path and, with it, the identity (and hence the location) of the issuer of that credential. That issuer is the subject of the next credential. By visiting each successive entity in the path and requesting their credentials, each credential in the path can be obtained, without broadcast.

A *backward traversable* path is analogous to a forward traversable path, except the credentials involved are held by issuers. A path $D \overset{*}{\to} A.r$ that is backward traversable can be discovered by doing backward search starting from $A.r$. Credentials involved in the path can be collected from entities starting with A and working from issuers to subjects.

Roughly speaking, a *confluent* path can be decomposed into two subpaths, one forward traversable and the other backward traversable. When both ends are known, a confluent path can be collected and assembled by starting at both ends and working inwards.

We define these notions of traversability for both edges and paths in credential graphs. Following the definition, we discuss the intuition behind traversability of derived edges.

DEFINITION 2 (TRAVERSIBILITY AND CONFLUENCE). *Let $G_{\mathcal{C}} = \langle N_{\mathcal{C}}, E_{\mathcal{C}} \rangle$ be the credential graph for a given set of credentials, $\mathcal{C}$.*
An edge added by closure property 1 is:

Forward traversable if the credential it represents is held by each subject of the credential;
Backward traversable if the credential it represents is held by the issuer of the credential;
Confluent if it is forward or backward traversable.

A path $e_1 \overset{*}{\to} e_2$ is:

Forward traversable if it is empty ($e_1 = e_2$), or it consists entirely of forward traversable edges;
Backward traversable if it is empty, or it consists entirely of backward traversable edges;
Confluent if it is empty, or it can be decomposed into $e_1 \overset{*}{\to} e' \to e'' \overset{*}{\to} e_2$ where $e_1 \overset{*}{\to} e'$ is forward traversable, $e'' \overset{*}{\to} e_2$ is backward traversable, and $e' \to e''$ is confluent. Note that paths that are forward traversable or backward traversable are also confluent.

An edge added by closure property 2, $B.r_2 \to A.r_1.r_2$ is:

Forward traversable if the path it is derived from, $B \overset{*}{\to} A.r_1$, is forward traversable;
Backward traversable if $B \overset{*}{\to} A.r_1$ is backward traversable;
Confluent if $B \overset{*}{\to} A.r_1$ is confluent;

An edge added by closure property 3, $D \to f_1 \cap \cdots \cap f_k$ is :

Forward traversable if (a) *there exists an $\ell \in [1..k]$ with $D \overset{*}{\to} f_\ell$ forward traversable, and* (b) *for each $j \in [1..k]$, $D \overset{*}{\to} f_j$ is confluent;*
Backward traversable if (a) *there exists an $\ell \in [1..k]$ with $D \overset{*}{\to} f_j$ backward traversable, and* (b) *for each $j \in [1..k]$, $D \overset{*}{\to} f_j$ is confluent;*
Confluent if for each $j \in [1..k]$, $D \overset{*}{\to} f_j$ is confluent;

Here is why a derived edge of the form $B.r_2 \to A.r_1.r_2$ has the same traversability as the path that it is derived from. Suppose there is a forward traversable path $D \overset{*}{\to} B.r_2 \to A.r_1.r_2 \overset{*}{\to} A.r$. Starting at D, a search agent can traverse to $B.r_2$. From there, the agent knows B, which enables it to continue searching, traversing $B \overset{*}{\to} A.r_1$. Upon reaching $A.r_1$, the search agent has proven the existence of $B.r_2 \to A.r_1.r_2$. Additionally, it knows A, so it can continue forward search from $A.r_1.r_2$.

Now suppose there is a forward traversable path $D \overset{*}{\to} A.r$ that can be decomposed into $D \to f_1 \cap \cdots \cap f_k \to B.r_1 \overset{*}{\to} A.r$. The edge $f_1 \cap \cdots \cap f_k \to B.r_1$ is forward traversable, so it is stored by the entity $base(f_j)$, for each $j \in [1..k]$. If there is one f_ℓ with $D \overset{*}{\to} f_\ell$ forward traversable, a search agent can use it to get from D to f_ℓ. From $base(f_\ell)$, the agent can obtain the credential $B.r_1 \longleftarrow f_1 \cap \cdots \cap f_k$, thereby identifying all other f_j's. The search agent then finds a path from

D to each f_j, and continues its forward search from $B.r_1$. Since both ends are known, each path $D \overset{*}{\rightarrow} f_j$ only needs to be confluent. The rationale for backward traversability of edges derived from backward traversible paths is similar.

4.2 A Credential Type System

If all credentials are stored by their issuers, all paths are backward traversable. Similarly, if all credentials are stored by their subjects, all paths are forward traversable. As we argued in section 1, neither arrangement by itself is satisfactory—greater flexibility is required in practice. Yet some constraints must be imposed on credential storage, or else many paths cannot be discovered. One way to organize those constraints is by requiring that all credentials defining a given role name have the same storage characteristics. Capitalizing on this observation to support distributed discovery, we introduce a type system for credential storage, the important feature of which is that, given a set of well-typed credentials, every path in its credential graph is confluent.

In our type system, each role name has two types: an issuer-side type specifies whether a search agent can trace credentials that define the role name by starting from the credentials' issuers; the other, a subject-side type, specifies these credentials' traceability from their subjects.

The possible issuer-side type values are *issuer-traces-none*, *issuer-traces-def*, and *issuer-traces-all*. If a role name r is issuer-traces-def, then from any entity A one can find all credentials defining $A.r$. In other words, A must store all credentials defining $A.r$. However, this does not guarantee that one can find all members of $A.r$. For instance, we might have $A.r \longleftarrow B.r_1$, with r_1 issuer-traces-none. This motivates the stronger type: issuer-traces-all. A role name r being issuer-traces-all implies not only that r is issuer-traces-def, but also that, for any entity A, using backward searching, one can find all the members of the role $A.r$.

The possible subject-side type values are *subject-traces-none* and *subject-traces-all*. If a role name r is subject-traces-all, then for any entity B, by using forward search, one can find all roles $A.r$ such that B is a member of $A.r$.

There are three values for the issuer-side type and two values for the subject-side type, yielding six combinations; however, a role name that is both issuer-traces-none and subject-traces-none is useless, so it is forbidden. This is captured by the notion of well-typedness.

We now extend this type system to role expressions and then define the notion of well-typed credentials. As we show in the next section, together these two definitions guarantee that when credentials are well-typed, the following three conditions hold. If a role expression e is issuer-traces-all, one can find all members of e by doing backward search from e. If e is subject-traces-all, then from any of its members, D, one can find a chain to e by doing forward search. If e is issuer-traces-def, then from any of its members, D, one can find a chain from D to e by doing bi-directional search.

Definition 3 (Types of role expressions).

- *A role expression is* well-typed *if it is not both issuer-traces-none and subject-traces-none.*
- *An entity A is both issuer-traces-all and subject-traces-all.*
- *A role $A.r$ has the same type as r.*
- *A linked role $A.r_1.r_2$ is*

$$\begin{cases} \textit{issuer-traces-all} & \textit{when both } r_1 \textit{ and } r_2 \textit{ are issuer-traces-all} \\ \textit{issuer-traces-def} & \textit{when } r_1 \textit{ is issuer-traces-all and } r_2 \textit{ is issuer-traces-def, or } r_1 \textit{ is issuer-traces-def and } r_2 \textit{ is subject-traces-all} \\ \textit{issuer-traces-none} & \textit{otherwise} \end{cases}$$

$$\begin{cases} \textit{subject-traces-all} & \textit{when both } r_1 \textit{ and } r_2 \textit{ are subject-traces-all} \\ \textit{subject-traces-none} & \textit{otherwise} \end{cases}$$

- *An intersection $f_1 \cap \cdots \cap f_k$ is*

$$\begin{cases} \textit{issuer-traces-all} & \textit{when there exists an } f_\ell \textit{ that is issuer-traces-all, and all } f_j\textit{'s are well-typed} \\ \textit{issuer-traces-def} & \textit{when all } f_j\textit{'s are well-typed} \\ \textit{issuer-traces-none} & \textit{otherwise} \end{cases}$$

$$\begin{cases} \textit{subject-traces-all} & \textit{when there exists an } f_\ell \textit{ that is subject-traces-all, and all } f_j\textit{'s are well-typed} \\ \textit{subject-traces-none} & \textit{otherwise} \end{cases}$$

The typing rule for a linked role $A.r_1.r_2$ may need some explanation. If both r_1 and r_2 are issuer-traces-all, then from $A.r_1.r_2$, one can find all members of $A.r_1$, and then, for each such member, B, find all members of $B.r_2$. If both r_1 and r_2 are subject-traces-all, then from any member, D, of $A.r_1.r_2$, one can first find that D is a member of $B.r_2$, and then find that B is a member of $A.r_1$, thereby determining that D is a member of $A.r_1.r_2$. Knowing both ends, D and $A.r_1.r_2$, one needs to find a middle point, $B.r_2$, using forward or backward search from one side. Then the other side can be handled by bi-direction search. If r_1 is issuer-traces-all, one can find all members of $A.r_1$, then r_2 only needs to be issuer-traces-def. Similarly, if r_2 is subject-traces-all, then one can trace to $B.r_2$ from D, and so r_1 only needs to be issuer-traces-def.

Definition 4 (Well-typed Credentials). *A credential $A.r \longleftarrow e$ is well-typed if all of the following conditions are satisfied:*

1. *Both $A.r$ and e are well typed.*
2. *If $A.r$ is issuer-traces-all, e must be issuer-traces-all.*
3. *If $A.r$ is subject-traces-all, e must be subject-traces-all.*
4. *If $A.r$ is issuer-traces-def or issuer-traces-all, A stores this credential.*
5. *If $A.r$ is subject-traces-all, every subject of this credential stores this credential.*

Consider credentials in example 3. One possible typing that makes all credentials well-typed is as follows: preferred, spdiscount, student, and university are issuer-traces-def, while accredited, stuID, and member are subject-traces-all.

4.3 Traversability with Well-typed Credentials

In this section we show that well-typed credentials whose storage is distributed can be located as needed to perform chain discovery.

LEMMA 6. *Assume $\mathcal{C}$ is a set of well-typed credentials and $G_{\mathcal{C}} = \langle N_{\mathcal{C}}, E_{\mathcal{C}} \rangle$ is the credential graph for $\mathcal{C}$. Let e be any role expression and D any entity. If there is a path $D \overset{*}{\to} e$ in $G_{\mathcal{C}}$, then we have the following:*

1. *$D \overset{*}{\to} e$ is confluent.*
2. *If e is issuer-traces-all, $D \overset{*}{\to} e$ is backward traversable.*
3. *If e is subject-traces-all, $D \overset{*}{\to} e$ is forward traversable.*

From Lemma 6 and Theorems 1 and 2, we have the following theorem, which says that if credentials are well typed, then role membership queries can be solved efficiently, even when credential storage is distributed. This is because confluent paths support efficient chain discovery, as discussed above in section 4.1. Furthermore, for roles of type issuer-traces-all, all members can be found efficiently. Finally, from any entity, it is possibly to find efficiently all subject-traces-all roles to which the entity belongs.

THEOREM 7. *Assume that $\mathcal{C}$ is a set of well-typed credentials and that $G_{\mathcal{C}} = \langle N_{\mathcal{C}}, E_{\mathcal{C}} \rangle$ is the credential graph for $\mathcal{C}$. Let A.r be any role and B any entity. Then we have the following:*

1. *$B \in \mathcal{S}_{\mathcal{C}}(A.r)$ if and only if there exists a confluent path $B \overset{*}{\to} A.r$ in $G_{\mathcal{C}}$.*
2. *If A.r is issuer-traces-all, then $B \in \mathcal{S}_{\mathcal{C}}(A.r)$ if and only if there exists a backward traversable path $B \overset{*}{\to} A.r$ in $G_{\mathcal{C}}$.*
3. *If A.r is subject-traces-all, then $B \in \mathcal{S}_{\mathcal{C}}(A.r)$ if and only if there exists a forward traversable path $B \overset{*}{\to} A.r$ in $G_{\mathcal{C}}$.*

Using a typing scheme such as the one presented here can also help improve the efficiency of centralized search, where type information can help choose nodes to be explored next.

4.4 Agreeing on Types and Role Meanings

Our type system begs the following question: How can entities agree on the type of a role name? This is the problem of establishing a common ontology (vocabulary), and it arises for RT_0 whether or not typing is introduced. Consider again the credentials in example 3. Given StateU.stuID ⟵ Alice, how does EPub know what StateU means by stuID? Is it issued to students registered in any class, or only to students enrolled in a degree program. This issue arises in all trust-management systems. Different entities need a common ontology before they can use each others' credentials. However, name agreement is particularly critical in systems, like RT_0, that support linked roles. For instance, the expression EOrg.university.stuID only makes sense when univerisities use stuID for the same purpose.

We achieve name agreement through a scheme inspired by XML namespaces [7]. One creates what we call *application domain specification documents* (ADSD), defining a suite of related role names. An ADSD gives the types of the role names it defines, as well as natural-language explanations of these role names, including the conditions under which credentials defining these role names should be issued. Credentials contain a preamble in which namespace identifiers are defined to refer to a particular ADSD, *e.g.*, by giving its URI. Each use of a role name inside the credential then incorporates such a namespace identifier as a prefix. Thus, a relatively short role name specifies a globally unique role name. Each ADSD defines a namespace. Note that this is a different level of namespaces from the notion of namespaces in SDSI. The latter concerns itself with who has the authority to define the members of a role; the former is about establishing common understandings of role names.

5. FUTURE AND RELATED WORK

In this section, we illustrate briefly the next step in our role-based trust-management language work. We then discuss other future directions and related work.

As mentioned in section 2, RT_0 is the first step in a series of role-based trust-management languages. We are extending the algorithms presented here to RT_1, where role names are terms with internal structure, including logical variables (whose notation starts with "?", as in ?file). For example, the credential OS.fileop(delete, ?file) ⟵ OS.owner(?file) can be used to express the policy that the operating system will let a file's owner delete the file. We are also working on defining an XML representation for RT_1 credentials and application domain specification documents, as we discussed in section 4.4. RT_1 will be reported in a forthcoming paper.

5.1 Typing and Complete Information

Inferencing based on distributed credentials is often limited by not knowing whether all relevant credentials are present. The standard solution to this problem is to limit the system to monotonic inference rules. This approach ensures that, even without access to all credentials, if the credentials that are present indicate D is a member of $A.r$, it is certainly true. Missing credentials could make you unable to prove D is a member of $A.r$, but cannot lead you to conclude D is a member of $A.r$ erroneously.

When credentials are well-typed, as defined here, this restriction to monotonic inference rules could be relaxed. The type system ensures we know who to contact to request the relevant credentials. So assuming they respond and we trust that they give us the credentials we ask for, we can assume that we obtain all the credentials that are relevant. In this context, it may be safe to use non-monotonic inference rules. This would allow, for instance, granting role membership contingent on not already being a member of another role. This could form a basis for supporting RBAC-style separation of duties, as well as negation as failure. It will be necessary to manage the trust issue. For instance, we may trust that some issuers will give us all relevant credentials, while not trusting some subjects to do the same.

5.2 Credential Sensitivity

Like most prior trust-management work, we assume here that credentials are freely available to the agent responsible for making access control decisions. In general, credentials may themselves be sensitive resources. Techniques have been introduced [18] that support credential exchange in a context where trust management is applied to credentials, as well as to more typical resources. (See [19] for additional references.) That work assumes that credential storage is centralized in two locations: with the resource requester and with the access mediator. It remains open to manage disclosure of sensitive credentials whose storage is distributed among the credential issuers and subjects.

5.3 Other Related Work

In section 2.1, we compared RT_0 credentials with name definition certificates in SDSI 2.0. In section 3.1 we reviewed existing work to chain discovery in SDSI. Now, we discuss some other related work.

QCM (Query Certificate Managers) [12] and QCM's variation SD3 [13] also address distributed credential discovery. The approach in QCM and SD3 assumes that issuer stores all credentials and every query is answered by doing backward searching. As we discussed in the introduction, this is impractical for many applications, including the one illustrated in example 3. Using backward search to determine whether Alice should get the discount requires one to begin by finding all ACM members and all university students.

Graph-based approaches to chain discovery have been used before, *e.g.*, by Aura [1] for SPKI delegation certificates and by Clarke *et al.* [8] for SDSI name certificates without linked names. Neither of them deals with linked names.

6. CONCLUSIONS

We have introduced a role-based trust-management language RT_0 and a set-theoretic semantics for it. We have also introduced credential graphs as a searchable representation of credentials in RT_0 and have proven that reachability in credential graphs is sound and complete with respect to the semantics of RT_0. Based on credential graphs, we have given goal-oriented algorithms to do credential chain discovery in RT_0. Because RT_0 is more expressive than SDSI, our algorithms can be used for chain discovery in SDSI, where existing algorithms in the literature either are not goal-oriented or require using specialized logic programming inferencing engines. Because our algorithms are goal-oriented, they can be used whether or not credentials are stored centrally. We have also introduced a type system for credential storage that guarantees distributed, well-typed credential chains can be discovered. This typing approach can be used for other trust-management systems as well.

7. ACKNOWLEDGEMENT

This work is supported by DARPA through AFRL/IF contract F30602-97-C-0336 and SPAWAR contracts N66001-00-C-8015 and N66001-01-C-8005. Sameer Ajmani made some helpful comments on an earlier version of this paper. We also thank anonymous reviewers for their helpful reports.

8. REFERENCES

[1] Tuomas Aura. Fast Access Control Decisions from Delegation Certificate Databases. In *Proceedings of 3rd Australasian Conference on Information Security and Privacy (ACISP '98)*, volume 1438 of *Lecture Note in Computer Science*, pages 284–295. Springer, 1998.

[2] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote Trust-Management System, Version 2. IETF RFC 2704, September 1999.

[3] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized Trust Management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, 1996.

[4] Matt Blaze, Joan Feigenbaum, and Martin Strauss. Compliance-Checking in the PolicyMaker Trust Management System. In *Proceedings of Second International Conference on Financial Cryptography (FC'98)*, volume 1465 of *Lecture Note in Computer Science*, pages 254–274. Springer, 1998.

[5] Sharon Boeyen, Tim Howes, and Patrick Richard. Internet X.509 Public Key Infrastructure LDAPc2 Schema. IETF RFC 2587, June 1999.

[6] Piero Bonatti and Pierangela Samarati. Regulating service access and information release on the web. In *Proceedings of the 7th ACM Computer and Communication Security*, pages 134–143, 2000.

[7] Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in XML. W3C Recommendation, January 1999. http://www.w3.org/TR/REC-xml-names/.

[8] Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certificate Chain Discovery in SPKI/SDSI. Manuscript submitted to Journal of Computer Security, December 2000. Available from http://theory.lcs.mit.edu/~rivest/publications.html.

[9] Yassir Elley, Anne Anderson, Steve Hanna, Sean Mullan, Radia Perlman, and Seth Proctor. Building Certificate Paths: Forward vs. Reverse. In *Proceedings of the 2001 Network and Distributed System Security Symposium (NDSS'01)*, pages 153–160. Internet Society, 2001.

[10] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI Certificate Theory. IETF RFC 2693, September 1999.

[11] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. Simple Public Key Certificates. Internet Draft (Work in Progress), July 1999. http://world.std.com/~cme/spki.txt.

[12] Carl A. Gunter and Trevor Jim. Policy-directed certificate retrieval. *Software: Practice & Experience*, 30(15):1609–1640, September 2000.

[13] Trevor Jim. SD3: a trust management system with certificate evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 106–115. IEEE Computer Society Press, 2001.

[14] Ninghui Li. Local Names in SPKI/SDSI. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW-13)*, pages 2–15. IEEE Computer Society Press, 2000.

[15] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. A Practically Implementable and Tractable Delegation Logic. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 27–42. IEEE Computer Society Press, 2000.

[16] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, February 1996.

[17] David S. Warren and *et al.* The XSB Programming System (Version 2.2), April 2000. http://www.cs.sunysb.edu/~sbprolog/xsb-page.html.

[18] William H. Winsborough, Kent E. Seamons, and Vicki E. Jones. Automated Trust Negotiation. In *DARPA Information Survivability Conference and Exposition*. IEEE Press, January 2000.

[19] T. Yu, M. Winslett, and K. E. Seamons. Interoperable strategies in automated trust negotiation. In *ACM Conference on Computer and Communications Security*, 2001.