



# Distributed credential chain discovery in $RT_0$

---

Authors: N. Li, W. H. Winsborough, J.  
C. Mitchell

Presenter: Ziad El Bizri

October 21, 2003

Purdue University - CS590U

1



## Outline

---

- $RT_0$ : a role-based trust-management language.
- Credential graphs
- Credential chain discovery algorithms
- Type system for distributed credential storage
- (Near) future work
- Conclusions

October 21, 2003

Purdue University - CS590U

2



## Trust management

---

- Goal: authorization in decentralized environments
- Basic tool: delegation of authority from one entity to others in the form of credentials.
- Access control decision: find a chain of credentials that delegates authority from the source to the requester.
- Previous TM systems: PolicyMaker, Keynote, SPKI/SDSI

October 21, 2003

Purdue University - CS590U

3



## Central issues

---

- Language for defining credentials
- Storage and retrieval of credentials in a distributed environment
- Flexible procedure for chain discovery that does not require all credentials at beginning of evaluation

October 21, 2003

Purdue University - CS590U

4



## RT<sub>0</sub>: a role-based trust-management language.

- Syntax:
  - Entities: Alice, EPub, ABU, StateU, ...
  - Role names: student, member, ...
  - Role: <Entity>.<Role name>
- Example:
  - ACM.member
  - StateU.student
- Semantics:
  - A role can be viewed as an attribute or as a group of entities members of the role

October 21, 2003

Purdue University - CS590U

5



## Credentials in RT<sub>0</sub> and semantics

- Simple member:
  - $A.r \leftarrow B$
  - $members[A.r] \supseteq \{B\}$
- Simple containment
  - $A.r \leftarrow B.r_1$
  - $members[A.r] \supseteq members[B.r_1]$
- Linking containment
  - $A.r \leftarrow A.r_1.r_2$
  - $members[A.r] \supseteq \cup_{B \in members[A.r_1]} members[B.r_2]$
- Intersection containment
  - $A.r \leftarrow B_1.r_1 \cap B_2.r_2 \cap \dots \cap B_n.r_n$
  - $members[A.r] \supseteq \cap_{1 \leq j \leq n} members[B_j.r_j]$
- A role expression  $e$  is either an entity, a role, a linked role or an intersection

October 21, 2003

Purdue University - CS590U

6



## Example

1. StateU.stuID  $\leftarrow$  Alice
  2. ABU.accredited  $\leftarrow$  StateU
  3. EPub.university  $\leftarrow$  ABU.accredited
  4. EPub.student  $\leftarrow$  EPub.university.stuID
  5. EPub.spdiscount  $\leftarrow$  EPub.student  $\cap$  EOrg.preferred
  6. EOrg.preferred  $\leftarrow$  ACM.member
  7. ACM.member  $\leftarrow$  Alice
- How do we show that  
 $members[EPub.spdiscount] \supseteq \{Alice\}$  ?

October 21, 2003

Purdue University - CS590U

7



## Query evaluation

- Three basic types of queries:
  - Given a role expression, find whether a specific subject is a member of it
  - Given a role expression, find all its members
  - Given a subject, find all roles it is a member of

October 21, 2003

Purdue University - CS590U

8



## Search strategies

- Forward (bottom-up): start with all credentials belonging to a requester and try to work up to the source
  - Useful for queries of the third type
- Backward (top-down): start with all credentials belonging to the source and try to work all the way down to the requester
  - Useful for queries of the second type
- Bidirectional: combine forward and backward strategies
  - Useful for queries of the first type

October 21, 2003

Purdue University - CS590U

9



## Credential and proof graphs

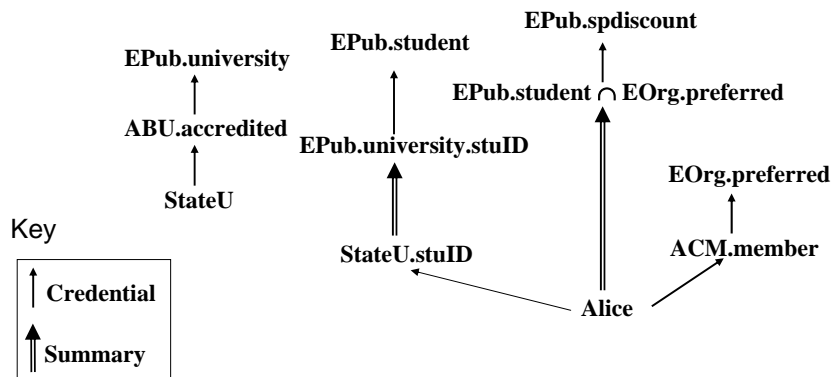
- A credential graph is a directed graph such that:
  - each *node* is a role expression
  - each *edge* represents a credential  $A.r \leftarrow e$
  - *Derived* edges are used to handle linked role and expressions
  - *Support* set of one or more paths are used to justify the existence of a derived edge
- A proof graph is a data structure that represents a credential graph and that maintains information about each node

October 21, 2003

Purdue University - CS590U

10

## Credential graph example



October 21, 2003

Purdue University - CS590U

11

## Graph-based search algorithm

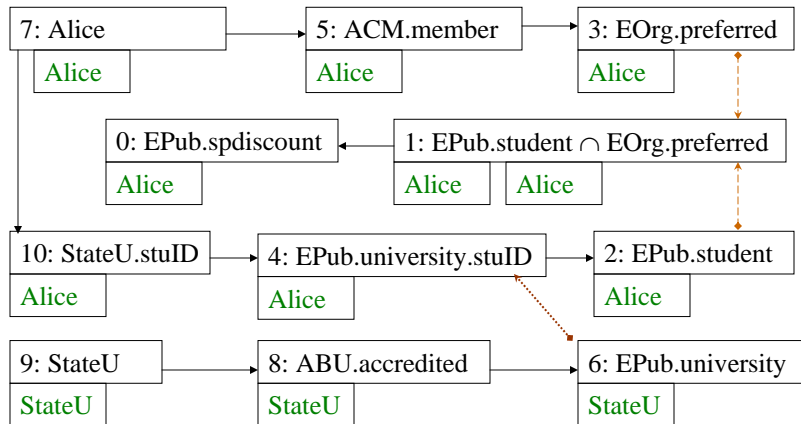
- Start with one node (role expression if backward, source if forward)
- Add edges and nodes one by one. If a new node is needed, add it to the queue of nodes to be processed.
- For each new node collect all its credentials and store all the role expressions it is a member of
- Repeat until a path is found between requester and source or until no edge or node can be added to the graph

October 21, 2003

Purdue University - CS590U

12

# Example of backward search algorithm

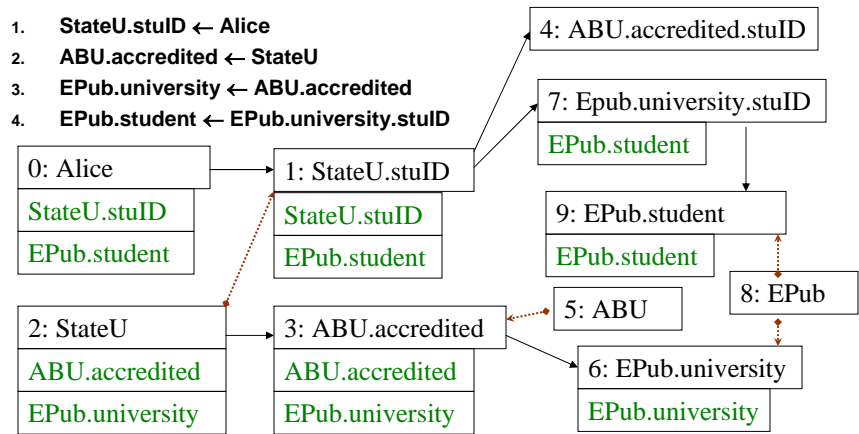


October 21, 2003

Purdue University - CS590U

13

# Example of forward search algorithm



1. StateU.stuID ← Alice
2. ABU.accredited ← StateU
3. EPub.university ← ABU.accredited
4. EPub.student ← EPub.university.stuID

October 21, 2003

Purdue University - CS590U

14



## Characteristics of proposed search algorithms

- Goal-directed
  - Backward search is goal-directed by nature
  - Forward search is goal directed since it starts by collecting only the requester's credential (not every possible credentials)
- Demand-driven
  - Credentials are collected only when needed
- Running time analysis: let  $N$  be the number of certificates and  $M$  be the sum of the sizes of all certificates ( $k$  for certificates defined as an intersection of  $k$  roles, 1 otherwise)
  - Backward search has a worst case running time of  $O(N^3+NM)$ , space  $O(NM)$
  - Forward search has a worst case running time of  $O(N^2M)$ , space  $O(NM)$

October 21, 2003

Purdue University - CS590U

15



## Bidirectional search properties

- Maintains two queues: one for forward search, and one for backward
- May construct proof graphs larger than forward search and backward search
- Main advantage: may succeed even when both methods would fail separately

October 21, 2003

Purdue University - CS590U

16



## Distributed credential storage

- The algorithms can be used to handle distributed credentials
- Difficulties arise as the credentials may be stored either with the issuer or the subject
  - Forward search cannot find credential if stored with issuer
  - Backward search cannot find credential if stored with subject
  - Bidirectional seems to be a better solution
  - What if  $A.r \leftarrow B.r_1$  is stored with A and  $B.r_1 \leftarrow C$  is stored with C, how do we find a path between C and A.r?
  - What if  $A.r \leftarrow B.r_1$  and  $B.r_1 \leftarrow C$  are both stored with B, how do we find a path between C and A.r?

October 21, 2003

Purdue University - CS590U

17



## Traversability and confluence

- Define forward traversable, backward traversable and confluent path. A credential edge is
  - forward traversable if it is stored by the subject of each credential
  - backward traversable if it is stored by the issuer of the credential
  - confluent if it is either forward or backward traversable
- Notion of traversability is easily extended to handle derived linked edges  $(A.r_1, r_2)$  and intersection edges  $(B_1.r_1 \cap \dots \cap B_n.r_n)$
- A path  $e_1 \rightarrow^* e_2$  is:
  - Forward traversable if it consists entirely of forward traversable edges
  - Backward traversable if it consists entirely of backward traversable edges
  - Confluent if it can be decomposed in  $e_1 \rightarrow^* e' \rightarrow e'' \rightarrow^* e_2$  where  $e_1 \rightarrow^* e'$  is forward traversable,  $e' \rightarrow e''$  is confluent and  $e'' \rightarrow^* e_2$  is backward traversable

October 21, 2003

Purdue University - CS590U

18



## Chain discovery

- Bi-directional distributed credential chain discovery finds all confluent paths.
- Q: How do we make sure that all legal paths are traversable?
- A: Introduce a type system for credentials!



## Type system for distributed credential storage

- Each role name  $r$  is assigned two types:
  - on the issuer side:
    - *issuer-traces-none*: the issuer of does not store any credential  $A.r$
    - *issuer-traces-def*: the issuer only stores the credential; it does not ensure that  $A$  can find all members of  $A.r$
    - *issuer-traces-all*: the issuer stores  $A.r$  and can find all its members (any  $A.r \leftarrow B.r_1$  has  $r_1$  as issuer-traces-all)
  - On the subject side:
    - *subject-traces-none*
    - *subject-traces-all*
  - Apply to the example  $A.r \leftarrow B.r_1$  and  $B.r_1 \leftarrow C$ : when can we link  $A.r$  to  $C$ ? Now what if we added subject-traces-def?



## Well typed role names

- A role name is
  - *Strongly well typed* if it is issuer-traces-all or subject-traces-all
  - *Weakly well typed* if it is both issuer-traces-def and subject-traces-none
  - *Well typed* if either strongly or weakly well typed
  - *Ill typed* otherwise (issuer-traces-none and subject-traces-none)

October 21, 2003

Purdue University - CS590U

21



## Types of role expressions

- An entity 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:
  - Issuer-traces-all if both  $r_1$  and  $r_2$  are issuer-traces-all
  - Subject-traces-all if both  $r_1$  and  $r_2$  are subject-traces-all
  - Weakly well typed otherwise, if either  $r_1$  is *issuer-traces-all* and  $r_2$  is well typed or  $r_1$  is well typed and  $r_2$  is subject-traces-all
  - Ill typed otherwise
- An intersection  $f_1 \cap \dots \cap f_k$  is:
  - Issuer-traces-all if there exists an  $f_j$  that is issuer-traces-all and remaining  $f_i$ 's are *well typed*
  - Subject-traces-all if there exists an  $f_j$  that is subject-traces-all and remaining  $f_i$ 's are *well typed*
  - Weakly well typed if all  $f_i$ 's are *weakly well typed*
  - Ill typed otherwise

October 21, 2003

Purdue University - CS590U

22



## Well typed credentials

- A credential  $A.r \leftarrow e$  is *structurally well typed* if the following three conditions are satisfied:
  - Both  $A.r$  and  $e$  are well typed
  - If  $A.r$  is issuer-traces-all,  $e$  must also be issuer-traces-all
  - If  $A.r$  is subject-traces-all,  $e$  must also be subject-traces-all
- A credential  $A.r \leftarrow e$  is *well typed* if it is structurally well typed and it satisfies the following two requirements:
  - If  $A.r$  is issuer-traces-def or issuer-traces-all,  $A$  stores this credential
  - If  $A.r$  is subject-traces-all, every subject of this credential stores this credential

October 21, 2003

Purdue University - CS590U

23



## Traversability of well typed chains

- Given an entity  $A$  and a role expression  $e$ :
  - If  $e$  is well typed (confluent), then one can use a bidirectional search algorithm to determine whether  $members[e] \supseteq \{A\}$
  - If  $e$  is issuer-traces-all (backward traversable), one can use a backward search algorithm to determine whether  $members[e] \supseteq \{A\}$
  - If  $e$  is subject-traces-all (forward traversable), one can use a forward search algorithm to determine whether  $members[e] \supseteq \{A\}$

October 21, 2003

Purdue University - CS590U

24

## Discussions

- The goal of the type system is to ensure that a path is always found when one exists
- It is a sufficient condition (and not a necessary condition) for traversability
- Handles traversability **and** storage requirements of credentials

October 21, 2003

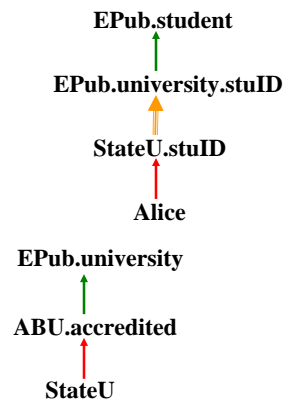
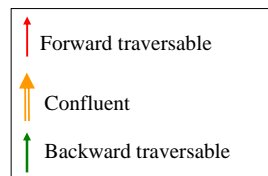
Purdue University - CS590U

25

## Example

- Types:
  1. student: issuer-def
  2. stuID: subject-all
  3. accredited: subject-all
  4. university: issuer-def

Key



October 21, 2003

Purdue University - CS590U

26



## Agreeing on typed and meanings of role names

- Credentials are issued by separate parties, however similar role names must be semantically identical
- Introduce *application domain specification documents (ADSDs)*
  - Specify vocabulary: role name, storage types and natural-language explanation of the role name
  - Can be extended to handle parametrized roles
  - A role is declared now in two parts: identifier for the ADSD that is used, identifier for the role declared inside the ADSD

October 21, 2003

Purdue University - CS590U

27




## Conclusion

- $RT_0$  – simple Role-based Trust-management language
- Credential graphs represent chains of delegation
- Proposed algorithms are efficient for centralized and distributed credential storage since they are goal-directed and collect credentials in a demand-driven way
- A type system is needed to enforce discovery of well typed chains in a distributed environment

October 21, 2003

Purdue University - CS590U

28



## (Near) future work

---

- Design similar algorithms to handle  $RT_1$  ( $RT_0$  with the inclusion of parameters)
- Issues:
  - Graphical methods cannot be applied
  - Logic programming search algorithms are needed



## Thank you

---

Questions?