CS590U
Access Control: Theory and Practice

Lecture 15 (March 8)

Distributed Credential Chain Discovery in Trust Management

Review: An Example in SDSI 2.0

- SDSI Certificates
  - (K_C access ⇨ K_C mit faculty secretary)
  - (K_C mit ⇨ K_M)
  - (K_M faculty ⇨ K_{EECS} faculty)
  - (K_{EECS} faculty ⇨ K_{Rivest})
  - (K_{Rivest} secretary ⇨ K_{Rivest} alice)
  - (K_{Rivest} alice ⇨ K_{Alice})
- From the above certificates, K_C concludes that K_{Alice} has access

Recap of the SDSI Rewriting-based Semantics

- Defines answers to queries having the form "can ω_1 rewrite into ω_2?"
- Specialized algorithms (either developed for SDSI or for model checking pushdown systems) are needed
- Papers by Abadi and Halpern and van der Meyden try to come up with axiom systems for the rewriting semantics

Defining Set-based Semantics (1)

- A valuation *V* maps each local name to a set of principals
- A valuation *V* can be extended to map each name string to a set of principals
  - $\underline{V}(K) = \{ K \}$
  - $\underline{V}(K\ A) = V(K\ A)$
  - $\underline{V}(K\ B_1 \ldots B_m) = \underset{j = 1..n}{\text{È}} \ \underline{V}(K_j\ B_2 \ldots B_m)$
    - where m>1 and $V(K\ B_1) = \{K_1, K_2, \ldots, K_n\}$

Defining Set-based Semantics (2)

- A 4-tuple (K A ⇨ ω) is the following constraint
  - V (K A) $\supseteq$ $\underline{V}$ (ω)
- The semantics of a set P of 4-tuples is the least valuation $V_{\mathbf{P}}$ that satisfies all the constraints
- Queries
  - "can ω rewrite into K?" answered by checking whether "K $\in$ $\underline{V}_{\mathbf{P}}$ (ω)".
- Does not define answers to "can ω_1 rewrite into ω_2".
  - asking whether $\underline{V}_{\mathbf{P}}$ (ω_1) $\supseteq$ $\underline{V}_{\mathbf{P}}$ (ω_2) is incorrect

Relationship Between Rewriting and Set Semantics

- Theorem: Given P, ω_1, and ω_2, ω_1 rewrites into ω_2 using P if and only if for any P' $\supseteq$ P, $\underline{V}_{\mathbf{P'}}(\omega_1) \supseteq \underline{V}_{\mathbf{P'}}(\omega_2)$.

- Corollary: Given P, ω, and K, ω rewrites into K using P if and only if $\underline{V}_{\mathbf{P}}(\omega) \supseteq \{ K \}$

What is RT?

- RT is a family of Role-based Trust-management languages
- Publications on RT
  - Li, Winsborough & Mitchell: “Distributed Credential Chain Discovery in Trust Management”, JCS’01, CCS’01
  - Li, Mitchell & Winsborough: “Design of a Role-Based Trust Management Framework”, S&P’02
  - Li & Mitchell: “Datalog with Constraints: A Foundation for Trust Management Languages”, PADL’03
  - Li & Mitchell: “RT: A Role-based Trust-management Framework”, DISCEX’03
  - Li, Winsborough & Mitchell: “Beyond Proof-of-compliance: Safety and Availability Analysis in Trust Management”, S&P’03

RT_0: An Example

1. **StateU.stuID ¬ Alice**
2. **ABU.accredited ¬ StateU**
3. **EPub.university ¬ ABU.accredited**
4. **EPub.student ¬ EPub.university.stuID**
5. **EPub.spdiscount ¬**
   EPub.student Ç EOrg.preferred
6. **EOrg.preferred ¬ ACM.member**
7. **ACM.member ¬ Alice**

- Together, the seven credentials prove that Alice is entitled to EPub's spdiscount

RT_0: Concepts and Credentials

- Concepts:
  - Entities (Principals): A, B, D
  - Role names: r, r_1, r_2 , …
  - Roles: $A.r$, $B.r_1$, … e.g., **`StateU.stuID`**
- Credentials: $A.r \leftarrow e$
  - Type-1: $A.r \leftarrow D$
  - Type-2: $A.r \leftarrow B.r_1$
  - Type-3: $A.r \leftarrow A.r_1.r_2$
    - **`e.g., EPub.student ¬ EPub.university.stuID`**
  - Type-4: $A.r \leftarrow B_1.r_1 \cap B_2.r_2 \cap \ldots \cap B_k.r_k$

RT_0 and SDSI 2.0

- SDSI 2.0 (The SDSI part of SPKI/SDSI 2.0)
  - has arbitrarily long linked names, e.g., $A.r_1.r_2.\ldots.r_k$, which can be broken up by introducing new role names
- RT_0
  - has intersection (type-4 credentials)
  - is thus more expressive than SDSI 2.0
    - algorithms for RT_0 can be used for SDSI 2.0

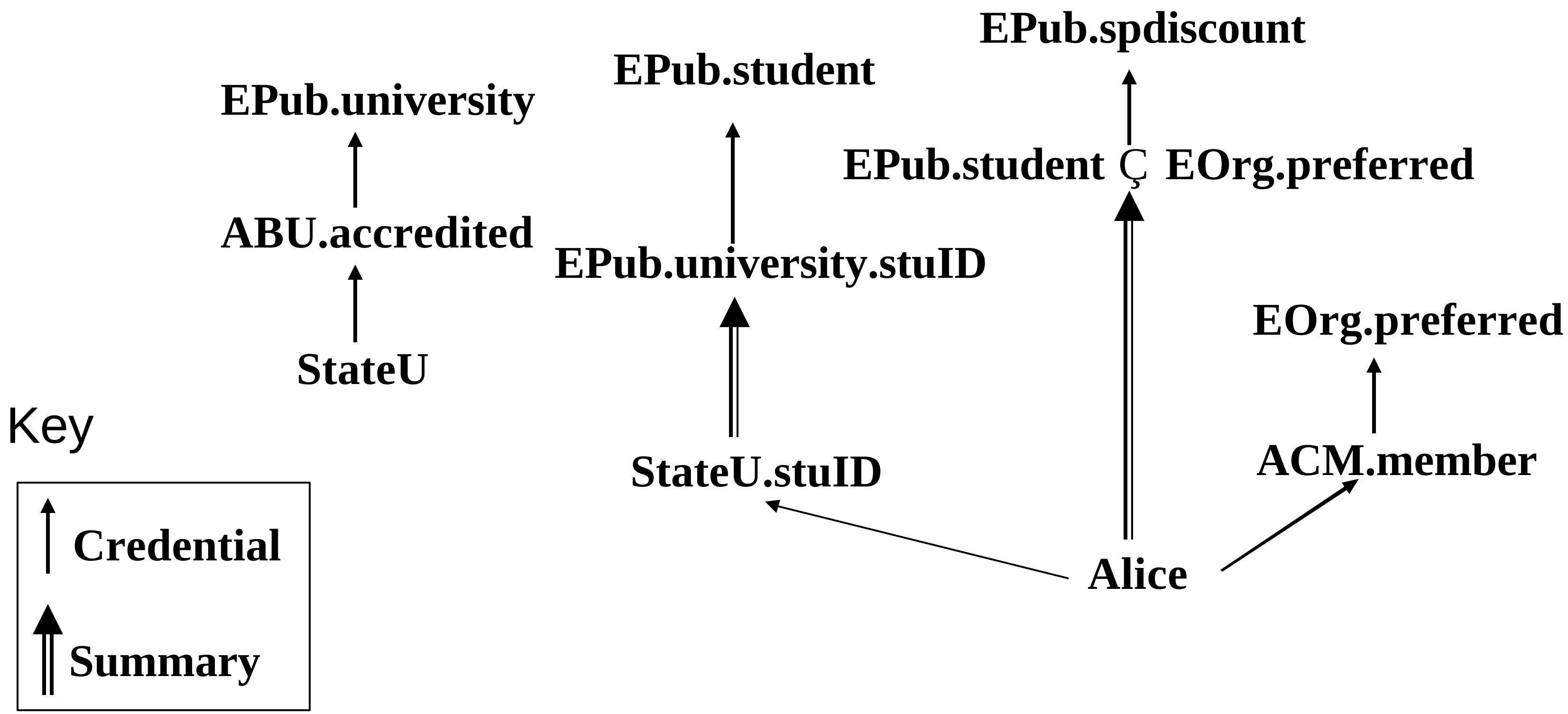
Goal-directed Chain Discovery

- Three kinds of queries and algorithms for answering them:
  1. Given A.r, determines its members
     - The backward search algorithm
  2. Given D, determines the set of roles that D is a member of
     - The forward search algorithm
  3. Given A.r and D, determines whether D is a member of A.r
     - The Bi-direction search algorithm

Credential Graph G_C

- Nodes:
  - A.r and *e* for each credential $A.r \leftarrow e$ in C
- Credential edges:
  - $e \rightarrow A.r$ for each credential $A.r \leftarrow e$ in C
- Summary edges:
  - $B.r_2 \rightarrow A.r_1.r_2$ if there is a path from B to $A.r_1$
  - $D \rightarrow A_1.r_1 \cap \ldots \cap A_k.r_k$
    if there are paths from D to each $A_j.r_j$
- Reachability in the credential graph is sound and complete wrt. the set semantics of RT_0

An Example Credential Graph

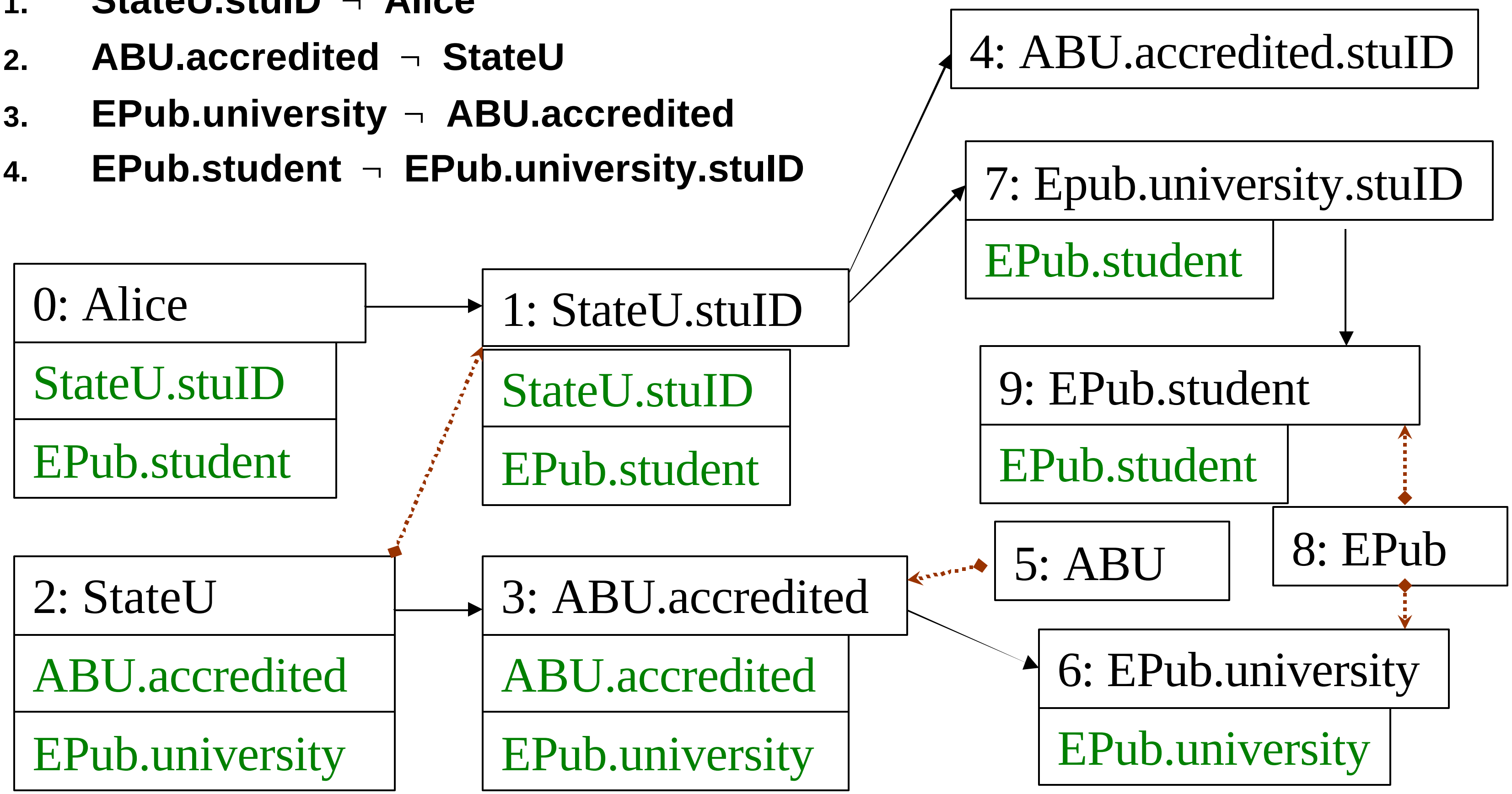


The Forward Search Algorithm (Overview)

- Starts with one entity node
- Constructs a proof graph
- Each node in the graph stores its solutions:
  - roles that this node can reach (is a member of)
- Maintains a work list of nodes need to be processed
- Algorithm Outline:
  - keep processing nodes in the work list until it is empty

Forward Search In Action

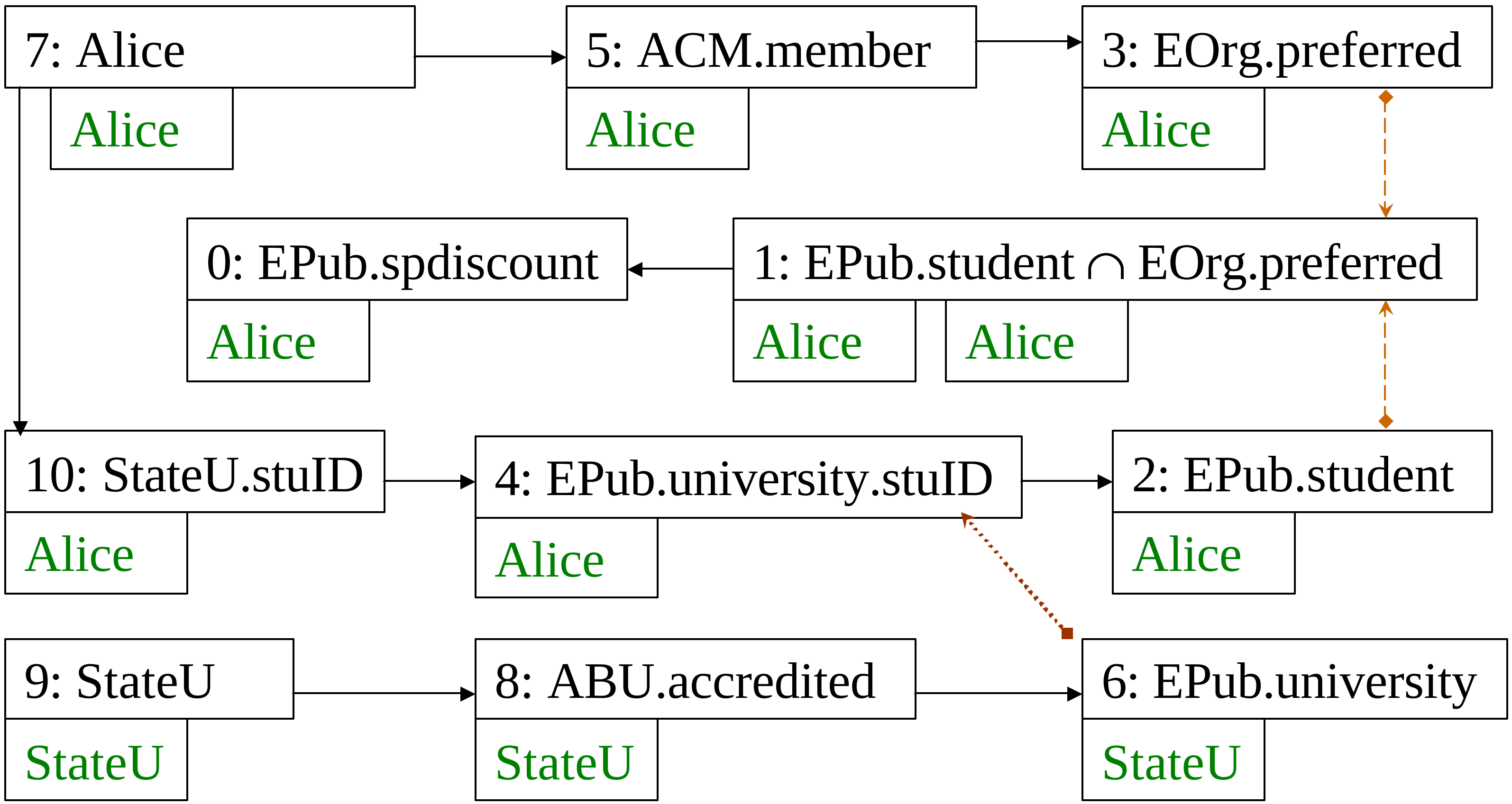
1. **StateU.stuID ← Alice**
2. **ABU.accredited ← StateU**
3. **EPub.university ← ABU.accredited**
4. **EPub.student ← EPub.university.stuID**



The Backward and Bi-direction Search Algorithms (Overview)

- The backward algorithm differs from the forward algorithm in that:
  - each node stores outgoing edges, instead of incoming ones
  - each node stores entities that can reach it, instead of roles that it can reach
  - the processing of a node is different
    - traversing the other direction
- The bi-direction search algorithm combines backward search and forward search

Backward Search In Action



Worst-Case Complexity

- Backward: time $O(N^3+NM)$, space $O(NM)$
  - N is the number of rules
  - M is the sum of the sizes of all rules,
    - $A.r \leftarrow f_1 \cap \ldots \cap f_k$ having size k, other credentials have size 1
- Forward: time $O(N^2M)$, space $O(NM)$
- However, this is goal oriented, making it much better in practice

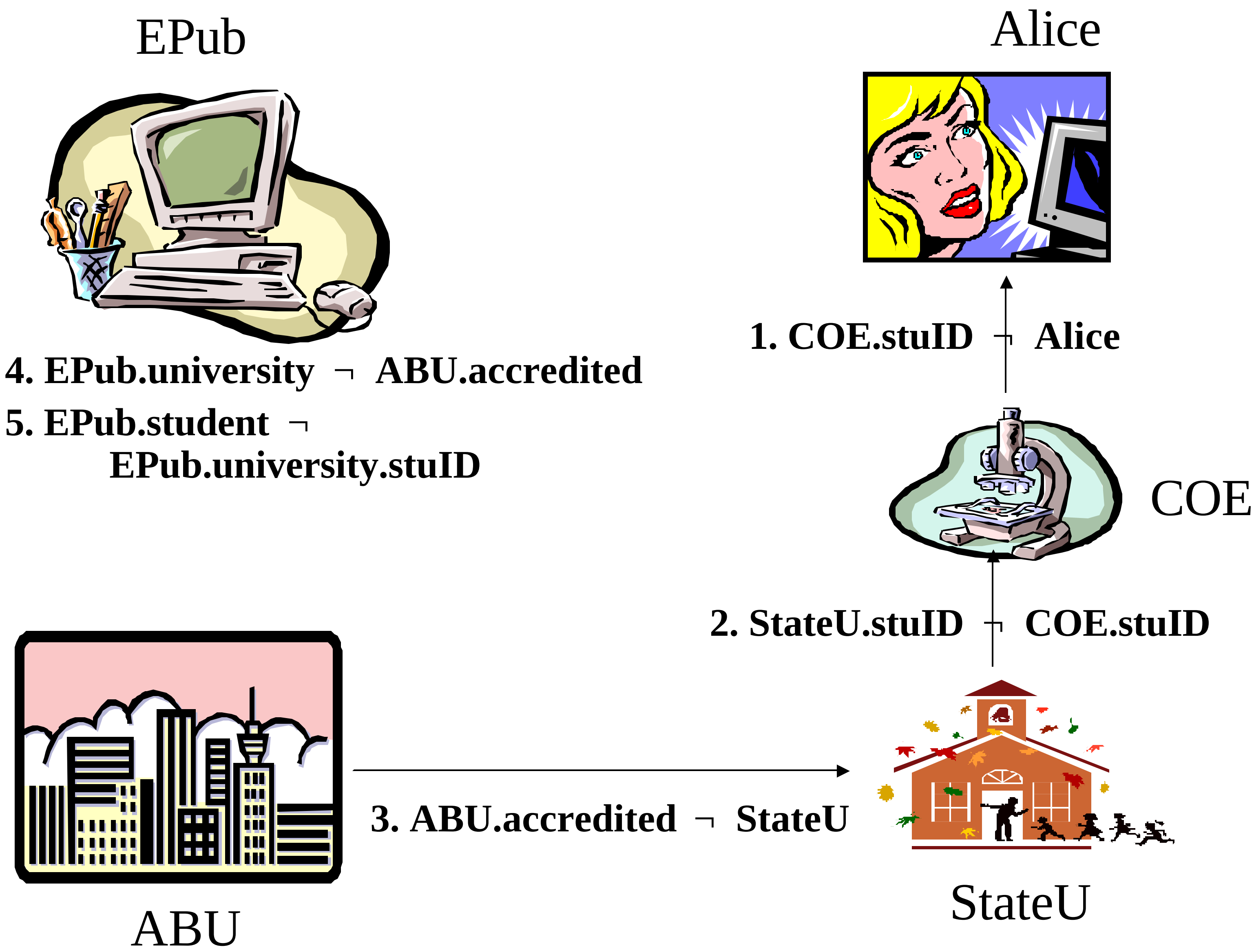
Why Develop These Algorithms?

- The queries can be answered using logic programs
  - however, this requires collection of all credentials in the system
- The backward algorithm is a goal-directed top-down algorithm
- The forward algorithm is a goal-directed bottom-up algorithm
- Distributed discovery requires combination of both

Distributed Storage of Credentials

- Example:
  1. EOrg.preferred ← ACM.member
  2. ACM.member ← Alice
- Who should store a credential?
  - either issuer or subject
- It is not reasonable to require that
  - all credentials are stored by issuers, or,
  - all are stored by subjects.

Who stores these statements?

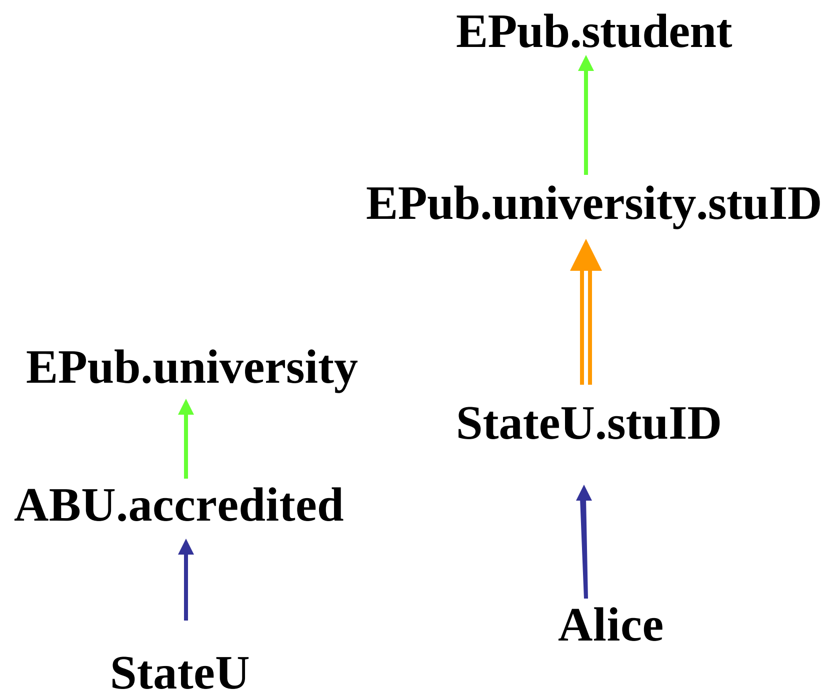


Traversability of Edges and Paths

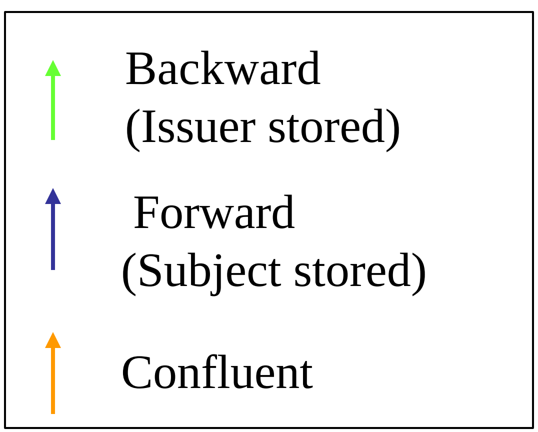
- A credential edge is
  - forward traversable, if stored by subject
  - backward traversable, if stored by issuer
  - confluent, if either forward traversable or backward traversable
- A path *e1* ➔ *e2* is
  - forward traversable, if all edges on it are, or *e1*=*e2*
  - backward traversable, if all edges on it are, or *e1*=*e2*
  - confluent, if it can be broken into *e1* ➔*e'* → *e''* ➔ *e2*,
    - With *e1*➔*e'* forward, *e'* → *e''* confluent, and *e''* ➔ *e2* backward

Traversability of Edges and Paths (con’d)

An edge $B.r_2 \rightarrow A.r_1.r_2$ has the same traversability as $B \Rightarrow A.r_1$



Key



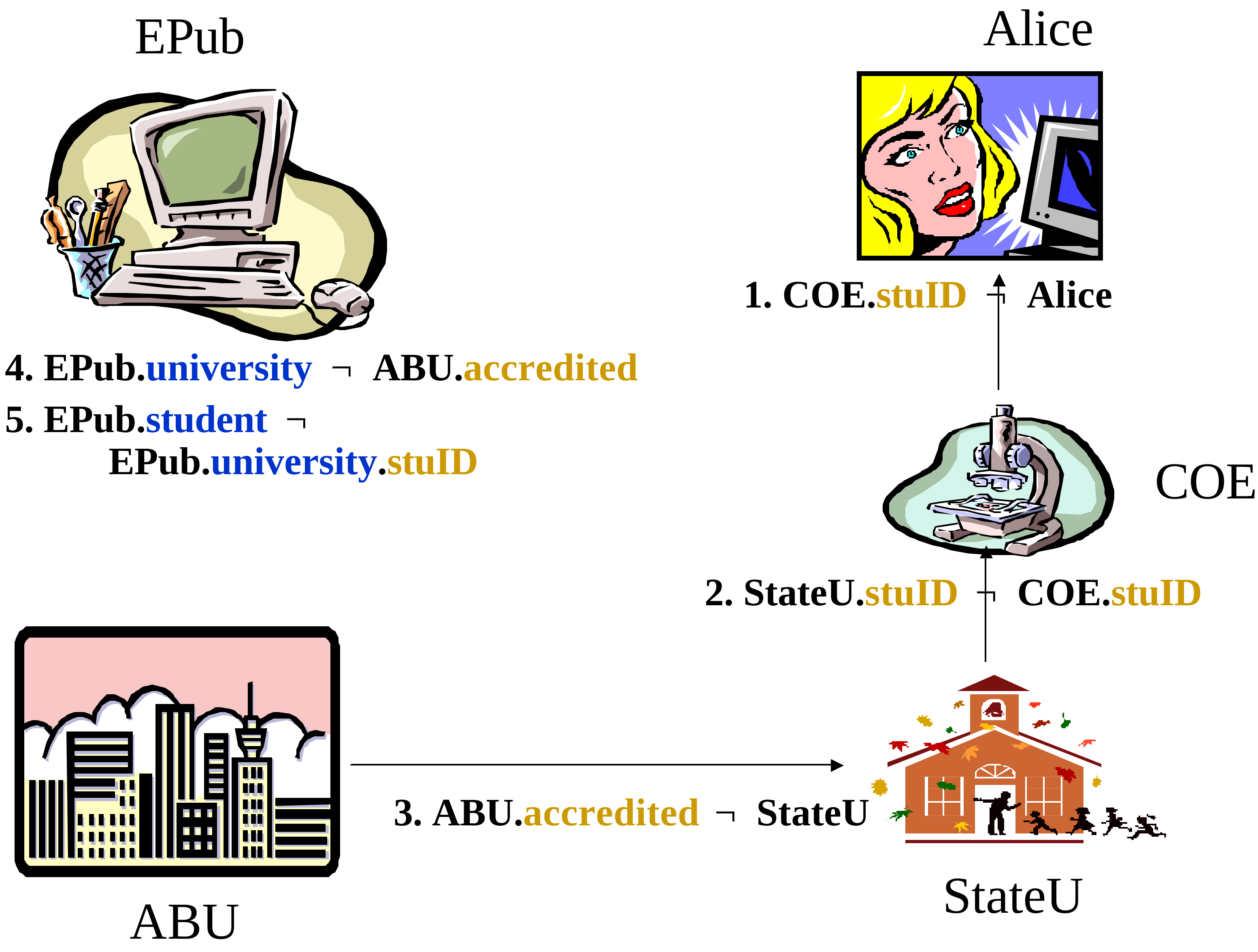
How to Ensure that Every Path is Confluent?

- Goal: using constraints local to each credential to ensure that every path is confluent
- Approach:
  - give each role name a traceability type
  - introduce a notion of well-typed credentials
- Main idea:
  - by requiring consistent storage strategy at role name level, we guarantee chains using well-typed credentials are confluent

Types of Role Names

- A role name has two types:
  - Issuer side:
    - issuer-traces-all
    - issuer-traces-def
    - issuer-traces-none
  - Subject side:
    - subject-traces-all
    - subject-traces-none

A Typing Scheme



Well-typed Credentials

- A credential A.r ← *e* is well-typed if :
  - Both A.r and *e* are well typed
    - A role A.r has the same type as r
    - A role expression is well-typed if it is **not** both issuer-none and subject-none
  - If A.r is issuer-def or issuer-all, then A must store the credential
  - If A.r is subject-all, then every subject of the credential must store it
  - If A.r is issuer-all, then *e* must be issuer-all
  - If A.r is subject-all, then *e* must be subject-all

Agreement on Types and Meaning of Role Names

- An approach inspired by XML namespaces
  - Use an Application Domain Specification Document (ADSD) to define a vocabulary
    - Each role has a storage type
  - Credentials have a preamble
    - Which defines vocabulary identifier to correspond to an ADSD
  - When using a role name, add a vocabulary identifier as prefix

Main Result about Type System

- Given a set of well-typed credentials $\mathbb{C}$, if D ➔ *e*
  - D ➔ e is confluent
  - if e is issuer-traces-all, D ➔ *e* is backward traversable
  - if e is subject-traces-all, D ➔ *e* is forward traversable

Benefits of the Storage Type System

- Guarantees that chains of well-typed credentials can be discovered
- Enables efficient chain discovery by telling the algorithm whether forward or backward search should be used for an intermediate query
- Communicates the application domain knowledge to the algorithm

Next Lecture

- More on SDSI Semantics and the RT Languages