Rule based Trust management using RT

Sandro Etalle

University of Twente – now on leave at the University of Trento

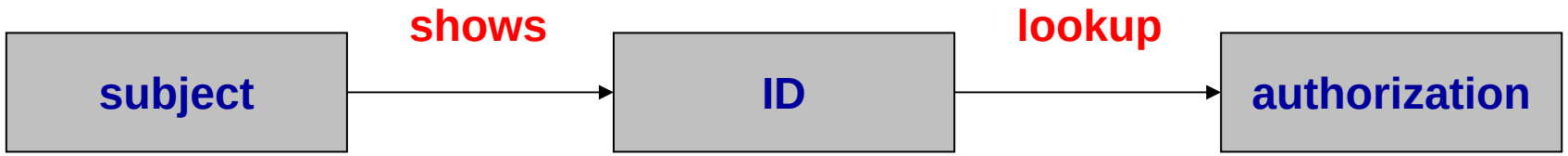
thanks to
Ninghui Li - Purdue
William H. Winsborough – University of Texas S. Antonio.
The DTM team of the UT (Marcin, Jeroen, Jerry).
And the many people I've taken ideas from for this lecture (Winslett, Li, Seamons…)

Overview

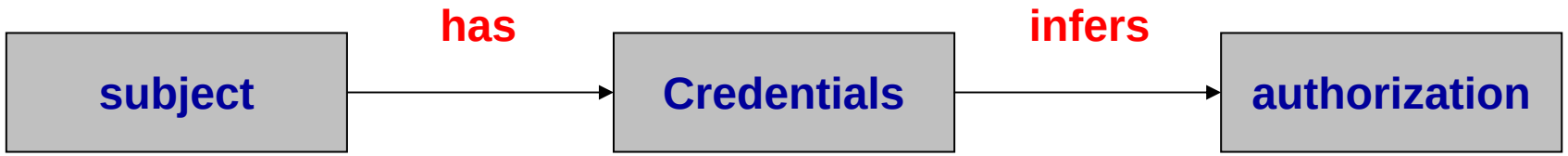
- Rule vs Reputation based TM
- A taste of trust negotiation
- The RT TM system
- Credential Chain discovery

Trust Management, the bottomline

- Typical access control mechanism



- TM alternative



2 flavors

- Reputation based TM
- Rule-based TM

Reputation-based TM *concrete*

- community of cooks (200 people)
- need to interact with someone you don't know,
  - to extablish trust:
    - you ask your friends
      - and friends of friends
        - ...
    - some recommendations are better than other
    - you check the record (if any)
  - after success trust increases

Reputation-based TM *virtual*

- p2p community of hackers (10000 people)
  - exchange programs & scripts
- need to interact with someone you don't know, ….
- difference with concrete community:
  - larger,
    - trust establishment has to be to some extent automatic

Reputation-based TM: salient features

- open system (different security domains)
- trust is a measure & changes in time
- essential risk component
- recommendation based (NOT identity-based)
- peers are not continuously available
- Some systems:
  - PGP,
  - EigenTrust Algorithm (Stanford)

rule-based TM: concrete example

- Bart is entitled to a discount

  If he is a student of the local university

rule-based tm, virtual

- When is Bart now entitled to a discount?

Bart is now entitled to a discount …

- If he is a student of any accredited University.
- But perhaps there are other reasons why Bart is entitled to a discount
  - If he is an employee of any governmental organization
  - If he is a member of the library club
  - If he is a veteran
  - ….
- Too many to mention
  - Which problems does this raise?
- Possible answers:
  - **Scalability**
  - **Knowing where and what to search**

Summary: reputation vs rules in TM

- open system (different security domains)
- trust is a measure & changes in time
- risk-based
- no delegation
- recommendation based (NOT identity-based)
- peers are not continuously available
- **scalability**

- open system (different security domains)
- trust is boolean & less time-dependent
- no risk
- delegation
- rule (credential) based (NOT identity-based)
- peers are not continuously available
- **scalability**

Bart is now entitled to a discount…(2)

- Bart wants to prove he is a student of an accredited university
  - He shows his GMU student ID
- But, is GMU *accredited?*
- *Accredited by whom?*
  - *The shop needs to specify this*
- How does Bart/the shop prove this?
  - By finding other credentials demonstrating this

Credential

- A credential is a *statement*
  - Signed by the *issuer*
  - about a *subject*
  - Containing info about the subject

- Requirements
  - Unforgeable (!)
  - Verifiable (that it belongs to the one asking for the service)
  - Signed (e.g. X509)
  - But most of all….
- **A well-defined semantics**

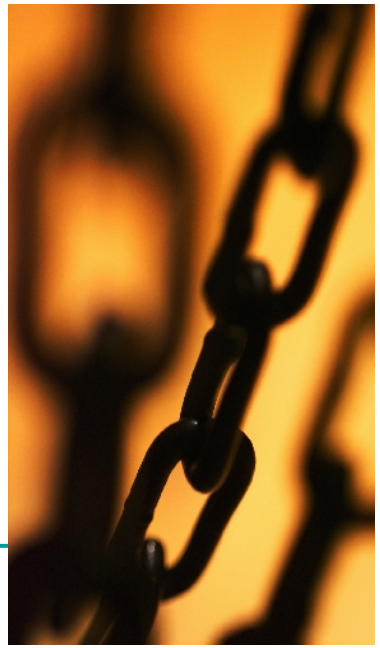
Bart is now entitled to a discount…(2)

- We have a *chain of credentials*
  - The subject of one is the issuer of the other one

2 features of rule based TM:

- No predefined security monitor
  - Needs a well-defined semantics
  - Credentials need to be disclosed to a *possibly untrusted party*
  - *ISSUE 1: Trust negotiation*

- Credentials are *distributed*
  - stored by the subject AND/OR by the issuer
  - *ISSUE 2: credential chain discovery*



A flavor of trust negotiation

- Credentials may contain private information and should be treated as such
  - E.g. medical record

Disclosing Credentials

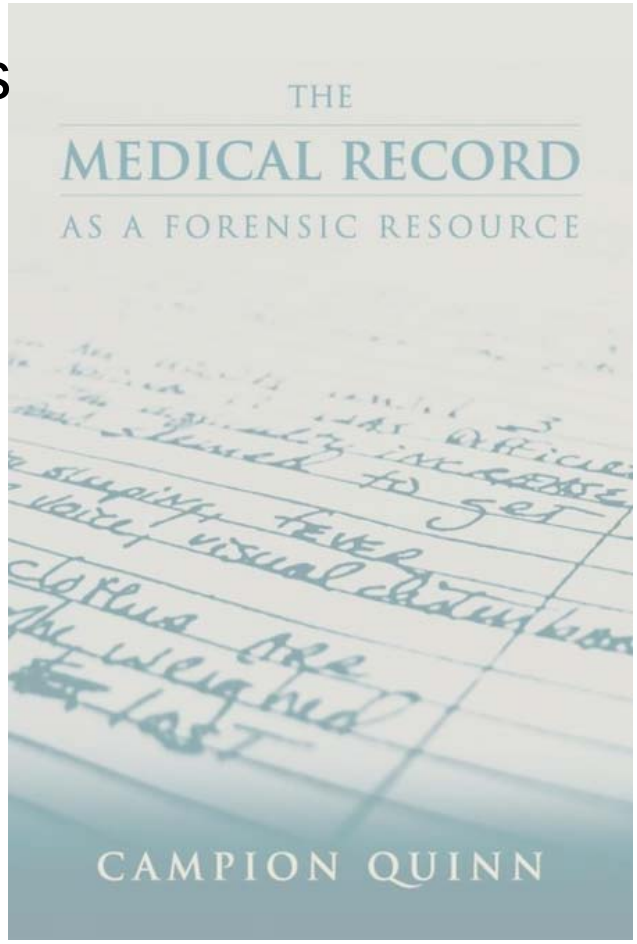
- Credentials should be disclosed only according to a given access control policy
- “I will show my medical record only to accredited surgeons”
- To *disclose a* credential one requires to see another credential

Example

- A: please send me this treatment (request)
- H: I'll do so if you show me your medical (policy)
- A: I'll show you my medical if you show me that you subscribe to GoodPrivacyPolicies
- H: Here is a credential showing this.
- A: here is my medical
- H here is the treatment.

Trust Negotiation

- Seamons: “The process of establishing trust between strangers in open systems based on the attributes of the participants”
- Goal: establish trust while maintaining privacy
- How: by iterative disclosure of credentials
- additional problem: what do you do with the info in a credential after it has been disclosed



Problems/challenges

- Many, to mention some:
- Circularities,
- Strategies (see [Seamons])
  - Naive
  - Reasonable
  - Informed

Question to think about

- *Clearly:* The disclosure of an additional credential may not lead to the revocation of a permission.
- But do we need full *monotonicity*?
- We are going to come back on this one…

Part 2

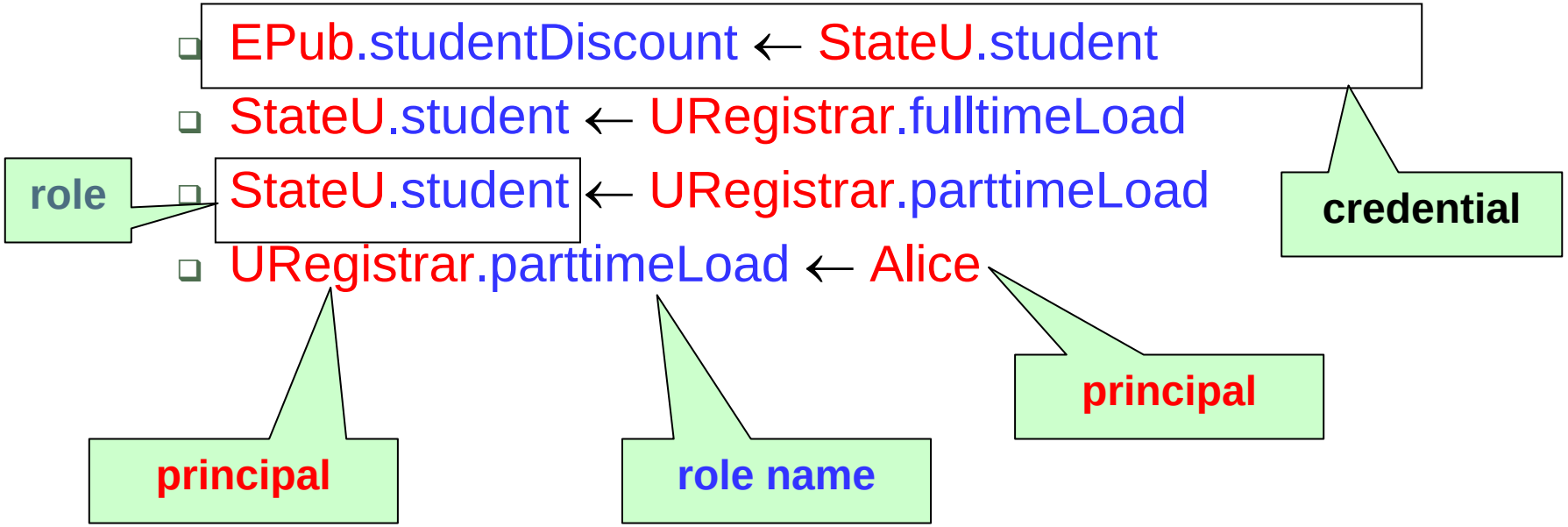
The RT family language

Policy Language Wish List

- Decentralize authority to define attributes
  - Utilize policy and credentials from many sources
- Delegation of attribute authority
  - To specific principals
  - To principals with certain attributes
- Intersection of attributes
- Parameterization, constraints
- Support for thresholds, separation of duty

Role-based Trust Management (*RT*)

- A family of credential / policy languages
  - Simplest, RT_0, has no parameterization, thresholds, or separation of duty [Li, Mitchell, Winsborough]
- RT_0 example: student discount subscription
  - EPub.studentDiscount ← StateU.student
  - StateU.student ← URegistrar.fulltimeLoad
  - StateU.student ← URegistrar.parttimeLoad
  - URegistrar.parttimeLoad ← Alice



RT0 Syntax

- A, B, D: principals
- r, r1, r2: role names
- A.r: a role (a principal + a role name)

- Four types of credentials:
  - $A.r \leftarrow D$
  - $A.r \leftarrow B.r1$
  - $A.r \leftarrow A.r1.r2$
  - $A.r \leftarrow A1.r1 \cap A2.r2$

Type 1 credentials

- Epub.discount $\leftarrow$ Alice
- Epub states that “Alice belongs to the role Epub.discount”
- Semantics Alice $\in$ [[Epub.discount]]
- Issuer: Epub
- Subject: Alice
- Where is this stored? We don’t know. Yet.
- Here I am trying to get away with something….

Type 2 credentials

- Epub.discount $\leftarrow$ StateU.student
- Epub states
  - "If StateU states that X is a student then I state that X gets a discount"
- Operationally:
  - "anyone showing a student certificate signed by stateU gets a discount"
- Epub *delegates* authority to StateU
- Semantics [[StateU.student]] $\subseteq$ [[Epub.discount]]
- Issuer: Epub
- Subject: StateU

Type 3 credentials

- Epub.discount ← AccredBureau.university.student
- Epub states
  - if AccredBureau states that X is an accredited university and
  - X states that Y is a student
  - then I state that Y gets a discount.
- “attribute-based delegation”
- Semantics
  - For every X ∈ [[AccredBureau.university]], [[X.student]] ⊆ [[Epub.discount]]
- Note:
  - like in SDSI, but links are of length max 2 (does not affect expressivity)
  - In the original RT0 the subject and the issuer are supposed to be the same

Type 4 credentials

- ITbizz.maysign $\leftarrow$ ITbizz.manager $\cap$ ITbizz.senior
- ITbizz states that "… senior managers may sign.."
  - "anyone showing a manager certificate and a senior certificate (both signed by ITbizz)) may 'sign'"
- Semantics
  - [[ITbizz.manager]] $\cap$[[ITbizz.senior]] $\subseteq$ [[ITbizz.maysign]]
- Issuer, subject: …

Summary:

- A, B, D: principals
- r, r1, r2: role names
- A.r: a role (a principal + a role name)

- Four types of credentials:

  - $A.r \leftarrow D$ Role A.r contains principal D as a member
  - $A.r \leftarrow B.r_1$ A.r contains role $B.r_1$ as a subset
  - $A.r \leftarrow A.r_1.r_2$ $A.r \supseteq B.r_2$ for each B in $A.r_1$
  - $A.r \leftarrow A_1.r_1 \cap A_2.r_2$ A.r contains the intersection

- The first 3 statement types: equivalent to pure SDSI
- Notice the higher-order flavour of $A.r \leftarrow A.r_1.r_2$.
- More complex versions have parameters (RT_1), constraints (RT_C), and can model thresholds and separation of duty (RT_T)

Exercise: find the semantics

- Alice.s ← Alice.u.v
- Alice.u ← Bob
- Bob.v ← Charlie
- Bob.v ← Charlie.s
- Charlie.s ← David
- Charlie.s ← Edward

Solution

- Alice.s ← Alice.u.v
- Alice.u ← Bob
- Bob.v ← Charlie
- Bob.v ← Charlie.s
- Charlie.s ← David
- Charlie.s ← Edward

- [[Charlie.s]] = {David, Edward}
- [[Bob.v]] = {Charlie, David, Edward}
- [[Alice.u]] = {Bob}
- [[Alice.s]] = {Charlie, David, Edward}

Other exercise

- The flexible company FC delegates the definition of buyer to any of its territorial divisions FCDiv1…. FCDivN
- FC uses the role FC.division to list all the territorial divisions.
- Accountants, on the other hand, must be approved by Accrinst, and must have a certification as controller given by FedCert.
- Alice is both a buyer and an accountant.
- Write an RT0 set of credentials for this.

Solution

- FC.division ← FCDiv1
- …
- FC.division ← FCDivN
- FC.buyer ← FC.division.buyer
- FCDiv1.buyer ← Alice
- FC.accountant ← Accrinnst.approved ∩ FedCert.controller
- Accrinst.approved ← Alice
- FedCert.controller ← Alice

Further down the lane

- In the Flexible Company FC, a buyer may also be an accountant, provided that his/her behaviour is logged.
- How do we do this?

First way of solving this

- Use negation in the policies
  - FC.acc ← FC.acc2 – FC.buyer
  - FC.acc ← FC.acc2 ∩ FC.buyer ∩ FC.log
  - FC.acc2 ← Accrinst.approved ∩ …
- But negation is nonmonotonic.
  - How do we deal with this?

A personal view on negation in TM.

- Negation is good provided that
  - It is always in a context
    - GOOD: all doctors that don't have a specialty
    - BAD: all non-doctors.
  - The negated predicate should rely on a definition we can "count on"
    - Eg: FC.acc ← FC.acc2 – FC.buyer
    - FC should be able to tell who populates FC.buyer without having to beg around for credentials.
- See paper by Czenko et. al
  - (yes, I confess, I am one of the authors).

A second way of solving this

- Using an integrity constraint.
- FC.log $\supseteq$ FC.buyer $\cap$ FC.accountant
- Need a mechanism to monitor it.
- External to the RT system.

  - See Etalle & Winsborough…

Conclusions

- Context:
  - 2 or more parties in an open system.
  - parties are not in the same security domain.
- Goal
  - establish trust between parties to exchange information and services (access control)
- Constraint
  - access control decision is made
    - NOT according to the party identity
    - BUT according to the credentials it has

Open problems

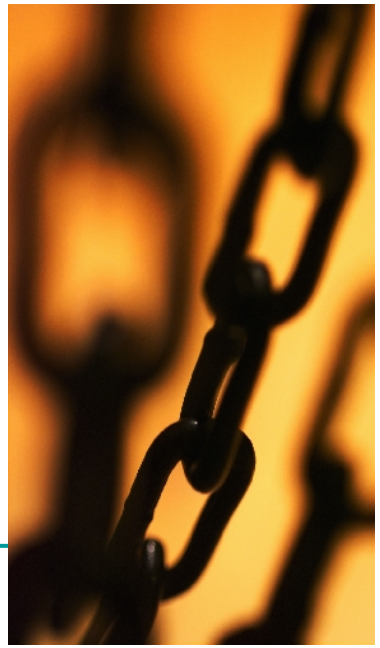
- Analysis
  - safety analysis
    - we are now working with Spin in RT0, for RTC (with constraints) nothing is available
  - of negotiations protocols w.r.t. the TM goals.
- Integration with other systems
  - e.g.
    - privacy protection
    - location-dependent policies
      - ambient calculi?
    - DRM
- Semantics
  - is not correct when considering:
    - chain discovery
    - negotiations
  - is not modular
  - certainly possible to improve this using previous work on omega-semantics.
- Types

Computing the trust

Memory refresh 1: Issues in TM

- No predefined security monitor
  - Needs a well-defined semantics
  - Credentials need to be disclosed to a *possibly untrusted party*
  - *ISSUE 1: Trust negotiation*

- Credentials are *distributed*
  - stored by the subject AND/OR by the issuer
  - *ISSUE 2: credential chain discovery*



Memory refresh (2): the old example

- Epub.discount ← Epub.accred.student
- Epub.accred ← ABU.accred
- ABU.accred ← GMU
- GMU.student ← Bart

- Query: “is Bart entitled to a discount?”
- Question: How do we proceed?
- Answer: it depends….
  - On WHAT?
  - On WHERE these credentials are stored.

Solution 1: store them by the *issuers*

- Then how we proceed?
- Epub asks the list of accredited universities to ABU.
- Epub asks to each of the accredited universities if Bart is a student….
- … did I mention we had a scalability problem?
  - It works, but it is not quite ideal.

Solution 2: store them by the *subjects*

- Epub asks Bart to show all his credentials.
- Bart has 1000 of them
- 900 of them are confidential,
  - so for these we have to start a trust negotiation.
- Afterwards, Epub asks to each of issuers of Bart's credentials if these credentials entail other credentials
- ... did I mention we had a scalability problem?
  - It works, but it is not quite ideal.

Solution 3: combine the two

1. Epub.discount ← Epub.accred.student
2. Epub.accred ← ABU.accred
3. ABU.accred ← GMU
4. GMU.student ← Bart

- Let Bart store #4, and the other creds by the issuers
- Ebu sees the first credentials and then asks Bart: “Are you a student at *any* university”?
- Bart shows his GMU credential.
- Epub checks that GMU is accredited (top down)
- DOESN’T ALWAYS WORK THIS NICELY…

So, where do we start from?

Queries

- Which kind of queries do we want to answer?

1. Given A.r and B, check if $B \in A.r$
2. Given A.r find out [[A.r]]
3. Given B, find out all A.r such that $B \in A.r$

- NB: [2] & [3] are more expressive than [1]
- Do we need all 3 of them?
- Do we need more?

First partial answer

- Suppose we try to get away just with type one query (B $\in$ A.r ?)
- In presence of the credential A.r $\leftarrow$ A.s.t
- … to answer the query B$\in$ A.r
- We need to compute ….
- [[A.s]]

➔ We need at least query types 1 and 2

➔ For the second partial answer we need to wait a bit.

Now, let's find an algorithm

- Query: B $\in$ A.r (B and A.r are given)
- Find a **top-down** algorithm for checking it.
  - Top-down: starting from A.r
- Take some time …
- Do we have a problem here?
- Yes: loops (just like in deductive DB)
  - A.r $\leftarrow$ C.s
  - A.r $\leftarrow$ …
  - C.s $\leftarrow$ A.r
  - C.s $\leftarrow$ …

Solving the loop problem

- Two ways:
- Bottom-up approach
- Top down + “loop checking”

Let's start with a bottom-up algoritm.

- Start: set [[A.r]] = { } for each A.r
  - Loop:
    - For each A.r ← D add D to [[A.r]]
    - For each A.r ← B.s add [[B.s]] to [[A.r]]
    - For each A.r ← A.s.t, for each B ∈ [[A.s]] add [[B.t]] to [[A.r]]
    - For each A.r ← B.s ∩ C.t …
  - Until nothing changes
- Summary
  - + simple
  - -- Not goal-directed (we *could* use magic sets)
  - -------- (!) We need the whole DB

Top-down algorithm, by example

- Example:
  - StateU.stud ← Alice
  - ABU.accredited ← StateU
  - EPub.university ← ABU.accredited
  - EPub.stud ← EPub.university.stud
  - EPub.discount ← EPub.stud ∩ EOrg.preferred
  - EOrg.preferred ← ACM.member
  - ACM.member ← Alice
- *Backward* search according to Li et. Al,
- Slide thanks to Li.

StateU.stud ← Alice

ABU.accredited ← StateU

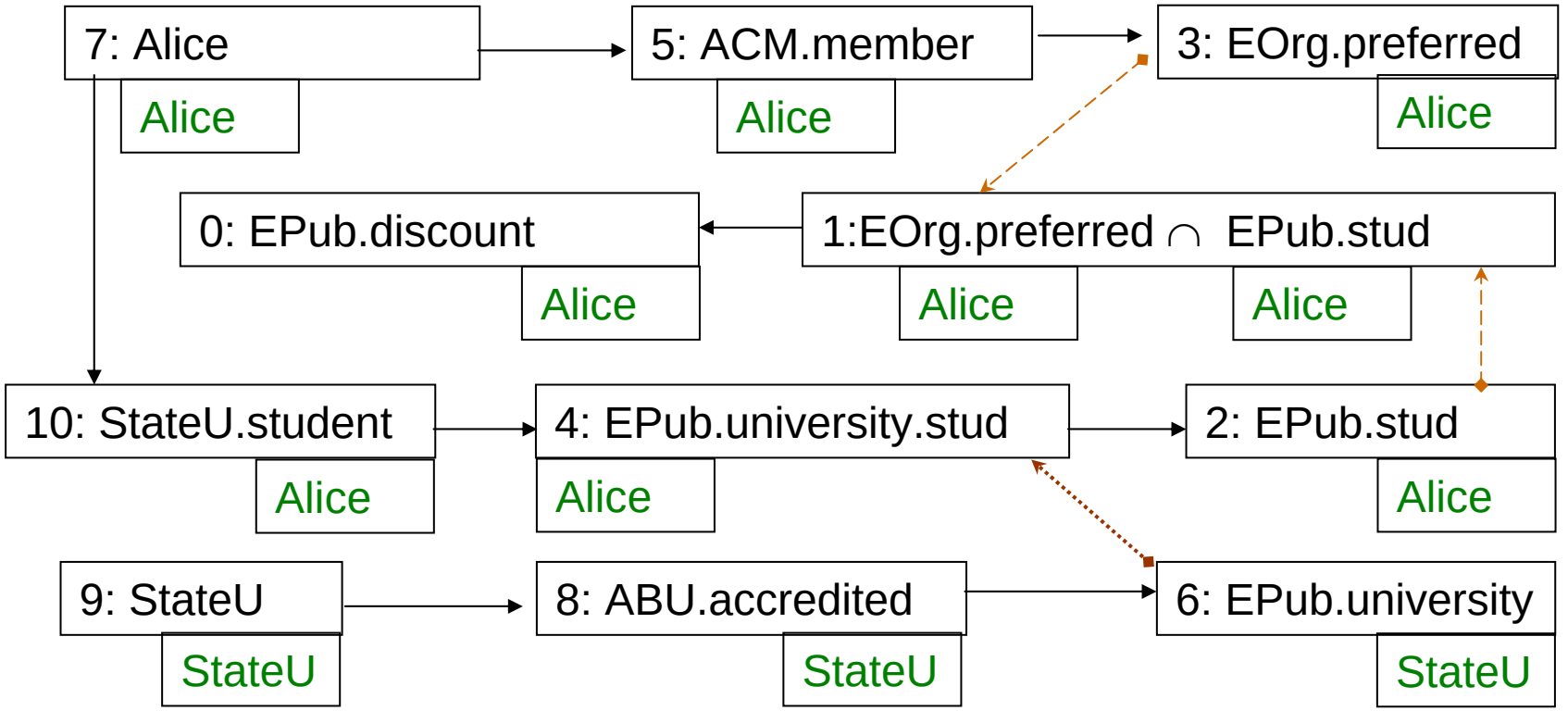
EPub.university ← ABU.accredited

EPub.stud ← EPub.university.stud

EPub.discount ← EPub.stud ∩ EOrg.preferred

EOrg.preferred ← ACM.member

ACM.member ← Alice



Top-down algorithm, summary

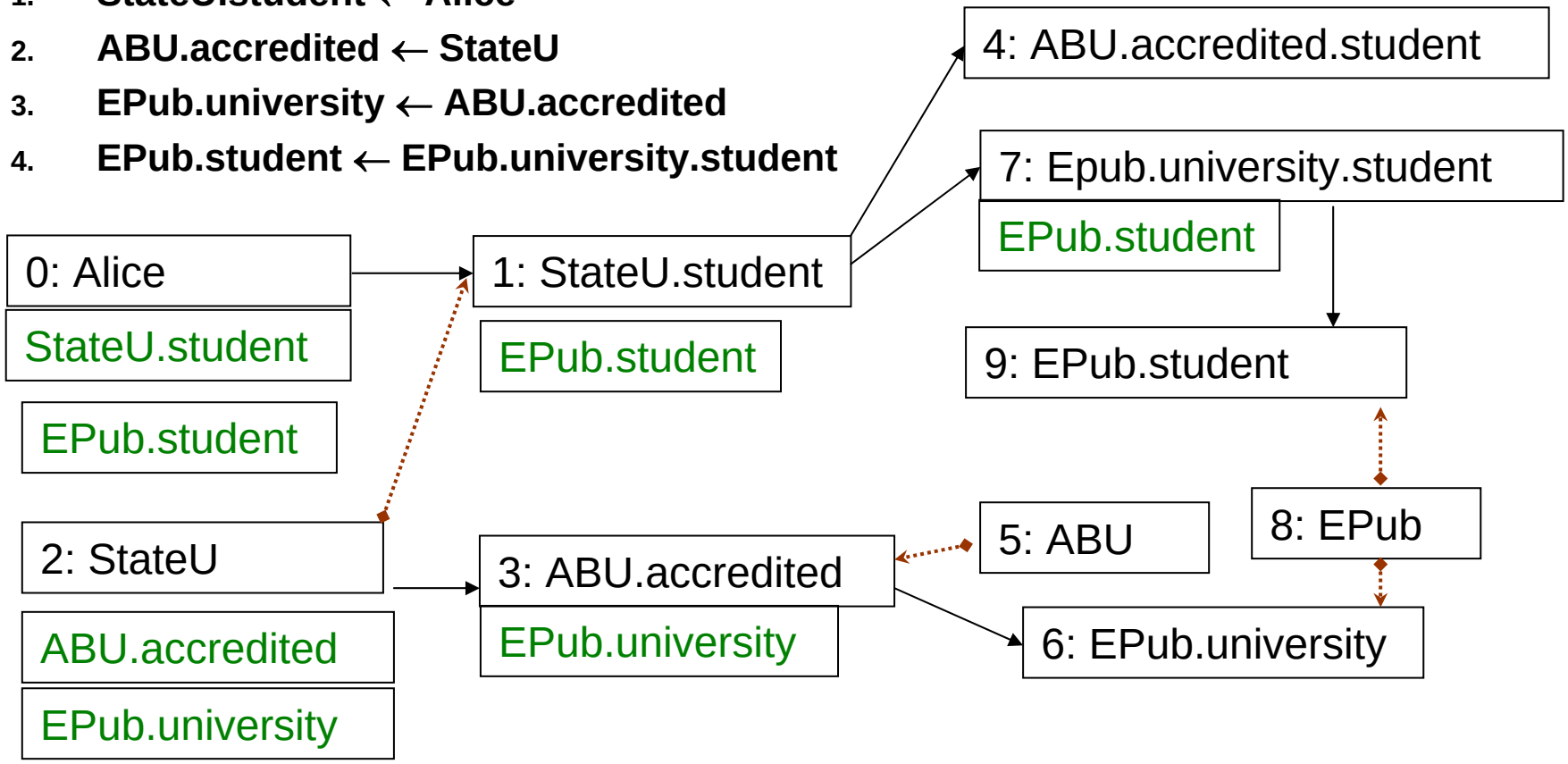
- - complex
- ++ goal directed
- ++ decentralized

Top-down algorithm, some further questions

- Who is doing the computation?
- Can he delegate part of the computation?
- Can we keep the credentials private?
- Where do we have to store the credentials?
- Answers:
  - The *role of the query* (EPub.spdiscount)
  - No, because of loops (A.r ← B.s, …, B.s ← A.r)
  - No, credentials have to be revealed (!)
  - By the issuer (!)

And if we would like to store the credentials by the subject?

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



"Forward Search" in Li's terminology. Thanks to Li for the slides

An issue with forward search

- Consider:
  - A.r ← B.r
  - B.r ← C.s
  - C.s ← Alice
  - D.t ← Alice
  - E.u ← Alice
  - F.v ← D.t
- Exercise:
  - check using forward search that Alice ∈ [[A.r]]

- What happens? (2)
- Alice must show/use all her credentials
  - Privacy, we had the dual problem in the backward search.
- We ave to compute a lot of useless credentials.
  - Also in the bacward search…
- One must answer queries like:
  - *Given B, find out all A.r such that B ∈ A.r*
  - Answers an old question of us

Combining Forward and Backward

- Why?
  - Forward needs credentials stored by issuers
  - Backward needs credentials stored by subjects
- We want to be able to store credentials
  - sometimes by issuers,
  - sometimes by subjects
  - Sometimes by both
- We need a combination of forward + backward search.
- What can go wrong?

What can go wrong

- Consider
  1. Alice.r ← Bob.s
  2. Bob.s ← Charlie
- Query: Charlie ∈ [[Alice.r]]
- Now, what happens if both credentials are stored by Bob?
- We cannot answer the query as we do not know where to start from.
- How many “situations” do we have?

Four situations

- Again
  1. Alice.r ← Bob.s
  2. Bob.s ← Charlie
- 4 situations
  - Both by issuer
  - Both by subject
  - 1. by issuer and 2. by subject
  - 1 by subject and 2. by issuer
- Which ones are OK?

Four situations, three types of queries

A. Given A.r and C, check if C $\in$ A.r
B. Given A.r find [[A.r]]
C. Given C, find out all roles R.t such that B $\in$ R.t

1. A.r $\leftarrow$ B.s
2. B.s $\leftarrow$ C

	queries		
storage	**A**	**B**	**C**
all by issuer	**OK**	**OK**	**NO**
all by subject	**OK**	**NO**	**OK**
(1) by issuer, (2) by subject	**OK**	**NO**	**NO**
(1) by subject, (2) by issuer	**NO**		

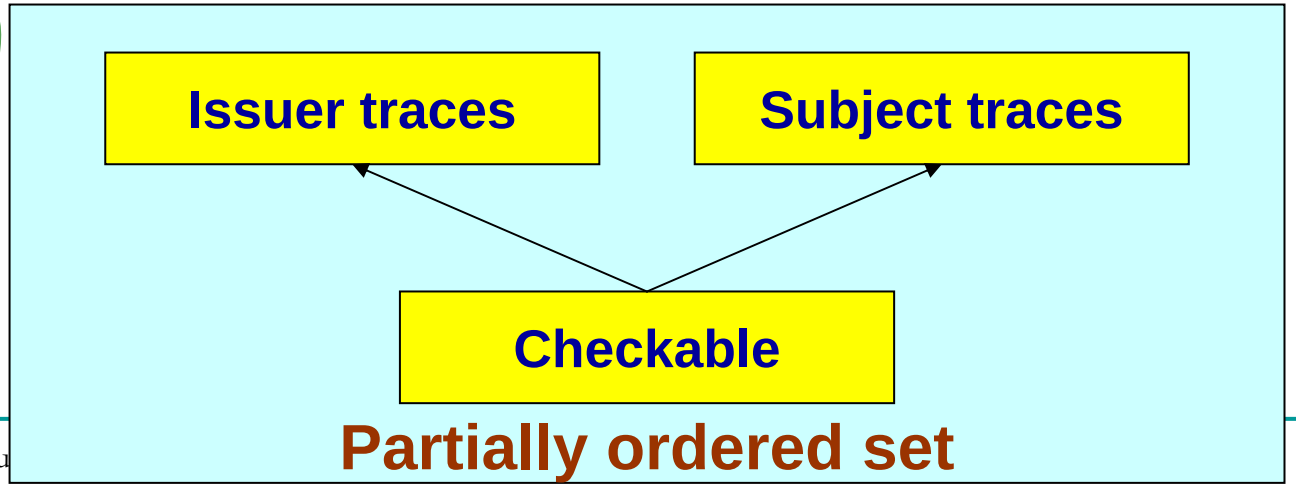
We need tools to

- Know which queries can be answered
- Rule out badly formed credentials
- …

Solution: a type system

- *Role names* have 3 types:
  - **Checkable** (officially issuer traces def)
    - For the query: given A.r and B, check if B $\in$ A.r
  - **Issuer traces**
    - For the query: given A.r, find [[A.r]]
  - **Subject traces**
    - For the query: given B, find out all A.r such that B $\in$ A.r

Partial answer



Well-typed credentials

Well typed credentials (1)

- If r is issuer traces
- A.r ← …
  - must be stored by A (the issuer)
- A.r ← B.s
  - s must be issuer traces
- A.r ← A.s.t
  - s and t must be issuer traces
- A.r ← B.s ∩ C.t
  - S must be issuer traces and t be well-typed (or vice versa)

Query: given A.r, find [[A.r]]

Well typed credentials (2)

- If r is subject traces
- A.r ← …
  - must be stored by the subject
- A.r ← B.s
  - s must be subject traces
- A.r ← A.s.t
  - s and t must be subject traces
- A.r ← B.s ∩ C.t
  - S must be subject traces and t be well-typed (or vice versa)

Query: given B,
find all A.r such that B ∈ A.r

Well typed credentials (3)

query: given A.r and B, check if B $\in$ A.r

- If r is checkable
- A.r $\leftarrow$ …
  - must be stored by the subject
- A.r $\leftarrow$ B.s
  - s must be well-typed (does not matter which type)
- A.r $\leftarrow$ A.s.t
  - Is well typed if
    - s is issuer trace & t is well-typed, OR
    - s isl well-typed and t is subject traces
- A.r $\leftarrow$ B.s $\cap$ C.t
  - s and t must be well-typed

One last thing

- A.r ← Charlie
- B.s ← Charlie

- r subject traceable and s issuer traceable.
- What is the answer to the query:
  - “Tell me all roles Charlie belongs to”.
- Answer is: {A.r}
- We miss B.s because s is not subject traceable (incompleteness)

Summarizing

- Unreasonable to think that all credentials be stored by subject (resp. issuer).
- The kind of queries we can answer to depends on the location of the credentials.
- Bad interplay of subject-stored and issuer-stored credentials can also prevent from finding the answer to a query.
- Types allow us to
  - Statically check when credentials are well-formed;
  - See which are the safe queries.

Biblio

- Ninghui Li, William H. Winsborough, John C. Mitchell: Distributed Credential Chain Discovery in Trust Management. Journal of Computer Security 11(1): 35-86 (2003)
- Ninghui Li, John C. Mitchell, William H. Winsborough: Design of a Role-Based Trust-Management Framework. IEEE Symposium on Security and Privacy 2002: 114-130
- Jim Basney, Wolfgang Nejdl, Daniel Olmedilla, Von Welch, Marianne Winslett: Negotiating Trust on the Grid. Semantic Grid 2005
- Marianne Winslett: An Introduction to Trust Negotiation. iTrust 2003: 275-283
- Ting Yu, Marianne Winslett, Kent E. Seamons: Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. ACM Trans. Inf. Syst. Secur. 6(1): 1-42 (2003)

Bonus track

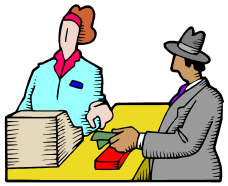
Integrity Constraints in TM

Why Integrity Constraints

- Policies do change: $P \Rightarrow P_1 \Rightarrow \ldots \Rightarrow P_n$
- A principal controls only a portion of the policy
  - Statements may be added or removed by other principals
  - nowadays: trusted principals give no feedback to the trusting ones
- Delegating trust implies an understanding between principals,
  - nowadays: not formalized
- Trusted principals need assistance in understanding global impact of delegations, revocations
  - Who could get access to what? (Safety)
    - Assessing exposure
  - Who could be denied? (Availability)
    - Ensuring applications have authorizations needed for correct operation

Problem Instances

- "No-one should ever be both a buyer and an accountant"
  - Mutual Exclusion



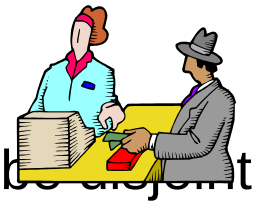
- "Welders of BOVAG-accredited workshops should be fellows of the British Institute of Welding"
  - Containment

- "Every employee should have access to the WLAN network"
  - Containment, Availability

Integrity Constraints: General Form

- General: $L.l \supseteq R.r$
  - $L.l \supseteq R.r$ holds in P iff $[[L.l]]_P \supseteq [[R.r]]_P$
  - L.l and R.r may be sets and intersections of roles
- Special cases
  - Membership: $A.r \supseteq \{ D_1, \ldots, D_n \}$
  - Boundedness: $\{ D_1, \ldots, D_n \} \supseteq A.r$
  - expressiveness is limited (it is a universal formula) but we can express all safety properties of [LWM03]
  - counterexample: at least a manager should have access to the DB

Examples



- buyers and accountants should be disjoint
  - $\varnothing \sqsupseteq A.buyer \cap A.accountant$
- every employee should have access to the WLAN network
  - $WLAN.access \sqsupseteq UT.employee$
- welders of BOVAG-accredited workshops should be fellows of the British Institute of Welding
  - Bovag.welder ← Bovag.accr.welder
  - Bovag.accr ← PietersWorkshop
  - PietersWorkshop.welder ← Pieter
  - $BIW.fellow \sqsupseteq Bovag.welder$

The technical problem

- P $\Rightarrow$ P1 $\Rightarrow$ ... $\Rightarrow$ Pn: policy change
- L.l $\sqsupseteq$ R.r: a constraint

- Need a (minimal) mechanism such that
  - IF L.l $\sqsupseteq$ R.r does not hold in Pi
  - THEN a warning is fired
  - without checking L.l $\sqsupseteq$ R.r each time a credential is added/removed

- How: by monitoring when some credentials are added or removed

The solution in short

- P – policy,
- Q = L.l ⊒ R.r – IC

- Define 2 set of roles:
  - G = roles R.r depends on
  - S = roles satisfying
    - $[[L.l]]_{P|S} \supseteq [[R.r]]_P$

- Theorem:
- Let P ⇒ P1 ⇒ ... ⇒ Pn
- IF
  - P satisfies Q
  - no credential S is removed
  - no credential for G is added
- THEN
  - Pn satisfies Q
  - G and S don't have to be recomputed

The method

- P – policy,
- Q = L.l ⊒ R.r: constraint

- CHECKING
- FIRST, compute $[[R.r]]_P$
  - here G is computed “for free”
- THEN, for each $X \in [[R.r]]_P$, check that $X \in [[L.l]]$
  - here (one of the) S is computed “for free”

- MONITORING
- Let P ⇒ P1 ⇒ ... ⇒ Pn
- IF
  - no credential for S is removed
  - no credential for G is added
- Then
  - OK
- Otherwise
  - Check Q again, and
  - Recompute G and S
    - (even if Q still holds)

To monitor this we need the cooperation of other principals

Extra Difficulty: non cooperating principals

- P $\Rightarrow$ P1 $\Rightarrow$ ... $\Rightarrow$ Pn: policy change
- L.l $\supseteq$ R.r: a constraint
- UT: set of untrusted principals
- P' is **reachable** from P iff diff(P,P') $\subseteq$ roles in UT

- Need a (minimal) mechanism such that
  - IF L.l $\supseteq$ R.r does not hold **in some P' reachable from P_i**
  - THEN a warning is fired
  - without checking L.l $\supseteq$ R.r each time

Dealing with non cooperating principals

- P – policy,
- Q = L.l ⊒ R.r: constraint
- T = trusted roles

- Define
  - UB(P) and LB(P)
    - new semantics [LMW04]
  - G = ... see paper ...
  - S = trusted roles such that
    - $[[L.l]]_{LB(P|S)} \supseteq [[R.r]]_{UB(P)}$

- Theorem:
- Let P ⇒ P1 ⇒ ... ⇒ Pn
- IF
  - $[[L.l]]_{LB(P)} \supseteq [[R.r]]_{UB(P)}$ and
  - no credential S is removed
  - no credential for G is added
- THEN
  - Every P' reachable from Pn satisfies Q
  - G and S don't have to be recomputed

Conclusions

- Integrity constraints:
  - tool to control a TM system.
- Monitoring requires the cooperation of trusted principals
- Trust management becomes a two way process
  - from the trusting to the trusted
  - and vice-versa