

Logic and Practice of Trust Management

Christian Skalka

Department of Computer Science
The University of Vermont

Joint work with X. Sean Wang (UVM), Peter Chapin (UVM), and Jeff Polakow (AIST, CVS, JST).

The Problem Setting

Modern distributed computing environments place unique demands on authorization for transactions and collaborations.

- *Decentralization*. No single, monolithic authority. Different domains comprise their own namespaces, beliefs, and policies.
- *Volatility*. Collaborations are dynamic, changing. Not all actors may be known to each other initially.
- *Non-trivial policies*. Usage policies may not simply be based on the identity of the requestor, but on a more complex set of conditions.

Simple access control does not suffice.

Trust Management Systems

Trust management systems (TMSs) provide a means to establish communications security between actors in this environment.

TMSs are *languages* for specifying and enforcing *authorization* in modern distributed systems. Recall the fundamental distinction between:

- *Authentication*: is this guy who he says he is?
- *Authorization*: is this guy allowed to use my stuff?

Authentication underlies authorization, but is usually transparent.

Examples of Trust Management Systems

- SPKI/SDSI. *Rivest, Lampson et al.*, SPKI Certificate Theory, *RFC-2693*, 1999.
- KeyNote. *Blaze, Feigenbaum et al.*, The KeyNote Trust-Management System Version 2, *RFC-2704*, 1999.
- Proof Carrying Authorization. *Lujo Bauer et al.*, A General and Flexible Access-Control System for the Web, *USENIX02*.
- *RT*. *Li, Mitchell, Winsborough*. Design of a Role Based Trust Management Framework, 2002 *IEEE Symposium on Security and Privacy*.

Components of Trust Management

While various TMSs differ significantly in their form and implementations, they share some common components:

- Language to express *credentials*, e.g. role membership, delegation authority.
- Language to specify local *policy*.
- *Semantics*, that specifies meaning of authorization.
- *Implementation*, that is sound with respect to the system semantics.

Policy and credential language are often the same. Semantics may not be decidable in general.

Trust Management in Practice

In addition to common theoretical underpinnings, all TMSs share some practical concerns:

- A strategy for collecting *certificates* that establish credentials in decentralized environment.
- Efficiency (decidability!) of authorization.
- *Verification* of system correctness: security is essential to get right.

The *RT* Framework

The *RT* framework is a family of trust management languages, with variants for delegation, constrained roles, and threshold policies.

- Each variant has as its core the RT_0 system (next slide).
- Policies and credentials have the same form.
- Each principal has a local namespace for roles (similar to SPKI/SDSI)
- Decidable semantics
- Liberal certificate distribution scheme.

The System RT_0

RT_0 credentials are constructed from *entities* (A, B, C) and *role names* (r, s, t), which are local to some entity ($A.r$). Credentials are of the form $A.r \longleftarrow f$.

1. $A.r \longleftarrow B$. *B is a member of $A.r$.*
2. $A.r \longleftarrow B.s$. *Any member of $B.s$ is a member of $A.r$.*
3. $A.r \longleftarrow B.s.t$. *If C is a member of $B.s$, then any member of $C.t$ is a member of $A.r$.*
4. $A.r \longleftarrow B_1.r_1 \cap B_2.r_2 \cap \dots \cap B_n.r_n$.
Any member of every $B_i.r_i$ is also a member of $A.r$.

Resources local to some A are represented as roles $A.r$, *authorization is determined by role membership* in $A.r$.

RT_0 Example

A hotel H wishes to offer discounts to its preferred customers and to members of certain organizations.

$$H.discount \leftarrow H.preferred \qquad H.discount \leftarrow H.orgs.members$$
$$H.orgs \leftarrow AAA$$

A later marketing decision by H adds $H.preferred \leftarrow AAA.members$.

Mary obtains credential $AAA.members \leftarrow M$. This authorizes Mary for the discount, i.e. establishes that M is in role $H.discount$ (in two different “ways”).

Why Logic is Good for TMSs

Mathematical logic is a natural setting for the specification and implementation of TMSs.

A rigorous, well-studied *mathematical foundation* for trust management, and the *right abstractions*.

- Formula language allows expression of credentials and sophisticated policies.
- *Authorization = logical validity*, given credential and policy assumptions.
- *Implementation = automated theorem proving*.
- *Verification = metatheory*, including properties such as soundness and consistency.

Previous Work

Both general and special purpose logics have long been used in application to distributed trust management (*authorization logics*).

- XSB Prolog, Datalog, Constraint Datalog used to specify SPKI/SDSI, Cassandra, *RT*.

Ninghui Li, Local Names in SPKI/SDSI, CSFW00

- Higher-order logic and Twelf serve as a foundation for PCA.

Lujo Bauer et al., A General and Flexible Access-Control System for the Web, USENIX02

- Delegation Logic.

Ninghui Li et al., A Logic-based Approach to Distributed Authorization, TISSEC02

A LolliMon Foundation for *RT*

We propose *LolliMon* as a foundation for trust management, specifically the *RT* framework.

LolliMon is a linear logic programming language. The most distinguishing feature of linear (vs. classical) logic can be summarized as:

In classical logic, a given fact can be used any number of times in a proof.

In linear logic, a given linear fact can be used only once in a proof.

LolliMon's *expressive formula language* and *type system* well-suited for application to TMSs (*RT* in particular).

Relevant LolliMon Features

Important details understood by simple comparison with Prolog/Datalog.

- $A :- B, C$ is written $A \leq B, C$ (can also write $B, C \Rightarrow A$).
- Linear implication $-o$, pronounced “lolli”, is analogue of \Rightarrow .
- The unrestricted modality $!A$ allows A to be treated classically.
- Formulae not restricted to Horn Clauses. Subgoals can be of a more general form, e.g. conditional subgoals in formulae: $A \leq (B \Rightarrow C)$.

LolliMon Proof Strategy

LolliMon uses a mix of forward chaining (bottom-up, Datalog style) strategy and backward chaining (top-down, Prolog style) strategy.

- The “membrane” separating regions of formulae proved by these strategies is called the monad $\{\dots\}^*$.
- Inside the monad, everything is proved using forward chaining strategy, and facts are linear by default.
- Outside the monad, everything is proved using a backward chaining strategy, and facts are classical by default.

*LolliMon: LOLLI + MONad

RT in Lollimon: Credentials Are Facts

Given the following constructors and type signatures:

```
a : entity.      b : entity.      r0 : role_name.  
                r1 : role_name.    r2 : role_name.
```

Credentials are denoted as classical facts.

$B.r_0 \longleftarrow A$ written as `credential b r0 (^ a)`.

$B.r_0 \longleftarrow A.r_1$ written as `credential b r0 (role a r1)`.

$B.r_0 \longleftarrow A.r_1.r_2$ written as `credential b r0 (linked_role a r1 r2)`.

Horn Clause Specification of RT_0

Role membership is specified as a predicate `ismem`. **NB**: also serves as implementation of role membership, correctness immediate.

```
ismem (role A R) B <= credential A R (^ B). // A.R <-- B
```

```
ismem (role A R0) D <=
  credential A R0 (role B R1), // A.R0 <-- B.R1
  ismem (role B R1) D.
```

```
ismem (role A R0) E <=
  credential A R0 (linked_role B R1 R2), // A.R0 <-- B.R1.R2
  ismem (role B R1) D,
  ismem (role D R2) E.
```

The Problem with Cyclic Credentials

A backward chaining specification causes nontermination in the presence of *cyclic credentials*, just as cyclic dependencies do in Prolog programs.

```
credential b r0 (role a r1).      credential a r1 (role b r0).
```

- Can be resolved using *tabling*, as in e.g. XSB Prolog.
- Can be resolved using forward chaining proof search, based on *saturation*.
- No tabling in LolliMon (yet?), so into the monad we go...

Forward Chaining Specification of RT_0

```
credential A R (^ B) => {!ismem (role A R) B}.
```

```
credential A R0 (role B R1),  
ismem (role B R1) D =>  
{!ismem (role A R0) D}.
```

```
credential A R0 (linked_role B R1 R2),  
ismem (role B R1) D,  
ismem (role D R2) E =>  
{!ismem (role A R0) E}.
```

Certificate Distribution and Retrieval

Trust management addresses the issues of decentralization and volatility by *distributing certificates*.

- Certificates held non-locally, establish credentials locally.
- Potentially enormous number of certificates, only a small fraction involved in authorization decision.

One sensible scheme is to require that the *requestor submit all certificates necessary for authorization*.

But, more flexible and practical schemes can be imagined...

Certificate Distribution

Returning to our example:

- $H.discount \leftarrow H.preferred$ and $H.discount \leftarrow H.orgs.members$ held by the hotel H .
- $H.orgs \leftarrow AAA$ and $H.preferred \leftarrow AAA.members$ held by AAA .
- $AAA.members \leftarrow M$ held by Mary.

Authorization must now be smart enough to *discover* the relevant certificates.

Discovery best driven by information encountered *during* authorization; problem called *certificate chain discovery*.

Certificate Chain Discovery

Two schemes for credential distribution proposed in *Li et al., Distributed Chain Discovery in Trust Management, JCS03*:

- *Issuer driven*: certificate for credential $A.r \leftarrow f$ held by A .
- *Subject driven*: certificate for credential $A.r \leftarrow f$ held by entity occurring in f .

For subject driven discovery, get certificates from subject, then proceed to issuers of those to continue chain.

1. Get $AAA.members \leftarrow M$ from Mary.
2. Get $H.preferred \leftarrow AAA.members$ from AAA.
3. Get $H.discount \leftarrow H.preferred$ from the Hotel H .

Certificate Chain Discovery

This is where previous logic-based approaches to *RT* stop.

Problem: Datalog and Prolog have no means of interleaving certificate discovery/retrieval and authorization steps in the logic.

- Authorization based solely on `ismem` presupposes that all certificates are given *a priori*.
- Horn clause logics have no way of dynamically adding new credentials (facts) to the set of assumptions.
- “Stopping” and “restarting” authorization with extended assumptions inelegant and impractical.

Certificate Chain Discovery

Previous solution for RT_0 : redefine semantics graph theoretically, write a Java program to implement.

- Correctness proof long and complicated.
- Not shown faithful to original Datalog semantics of RT_0 .
- *Not scalable to full RT .*

LolliMon solution easier to prove correct (with greater confidence), faithful to original RT semantics, easily scalable to RT variations.

Conditional Subgoals for Discovery

Intuition: *in logic, conditional statements $P \Rightarrow Q$ are proved by adding P to the environment of assumptions, and then proving Q .*

Assume given a predicate `retrieve` such that `retrieve A B Rb RE` holds iff a certificate establishing `credential B Rb RE` is retrievable from `A`.

```
auth R A <= ismem R A.  
auth R A <=  
  retrieve A B Rb RE,  
  (credential B Rb RE => auth R A).
```

New credentials are dynamically added upon discovery, as hypotheses.

Logic of Discovery

To construct a certificate chain, parameters of retrieval can be logically specified.

- Information about locations for future retrieval contained in existing credentials.
- LolliMon implementation interleaves retrieval and `ismem` steps.
 - Forward chaining proof context memoizes partial solutions.

In more detail: certificates are modeled as *linear* facts.

```
#linear certificate b r0 (linked_role a r1 r2).
```

- Linearity ensures no certificate is retrieved more than once.

Subject Driven Discovery

With details of retrieval left abstract:

```
credential A Ra Re,  
certificate B Rb (role A Ra) -o  
{!credential B Rb (role A Ra)}.
```

```
credential A Ra Re,  
certificate B Rb (^ A) -o  
{!credential B Rb (^ A)}.
```

```
credential A Ra Re,  
ismem (role D R) A,  
certificate B Rb (linked_role D R Ra) -o  
{!credential B Rb (linked_role D R Ra)}.
```

RT Framework Variations

Extensions to RT_0 in the *RT* framework:

- RT_1 and RT_2 add *constrained role name parameters* to RT_0 .

$StateU.foundingAlumni \leftarrow StateU.diploma(X, Y : [1955..1958])$

- RT^D adds delegation.
- RT^T adds threshold policies.

Usual abstractions of graph theory not applicable to these features.

To express in Lollimon, need only modify `ismem` appropriately, discovery technique unchanged (orthogonal to `ismem`). Correctness essentially immediate.

An Objection

Programming logics like LolliMon are not as efficient as C, or even Java.

If the majority of expense for distributed trust management is in the cost of authorization computation steps, objection sustained. But...

Hypothesis: the majority of expense for distributed trust management will lie in the cost of non-local certificate retrieval, with most decisions involving few credentials.

Being smart about certificate retrieval can be encoded in the logic of trust management.

Risk Averse Discovery

Being smart about retrieving certificates often implies taking *risk* into account.

- If risk means expected time of certificate retrieval, minimizing risk in discovery promotes efficiency.

Risk can be conceived more generally, for a wider range of benefits.

- Some certificates could be *assumed* to exist, with a higher level of risk.
- Some might be signed by questionable keys.
- Some might be near expiration.

The System RT^R

RT^R extends RT_0 by assigned risk values to credentials *Peter Chapin, Christian Skalka, X. Sean Wang, Risk Assessment in Distributed Authorization, FMSE05.*

- Credentials of the form $A.r \xleftarrow{\kappa} f$, where κ is some “risk”.
- Risks are ordered, can be combined (aggregated).
- Role membership parameterized by the aggregation of risks associated with relevant credentials.

Implementation parameterized by maximum tolerable risk *threshold* κ_M , as a heuristic for chain discovery.

Credential Risk Example

Letting risks be represented by integers:

$$H.discount \xleftarrow{5} H.preferred$$

$$H.discount \xleftarrow{5} H.orgs.members$$

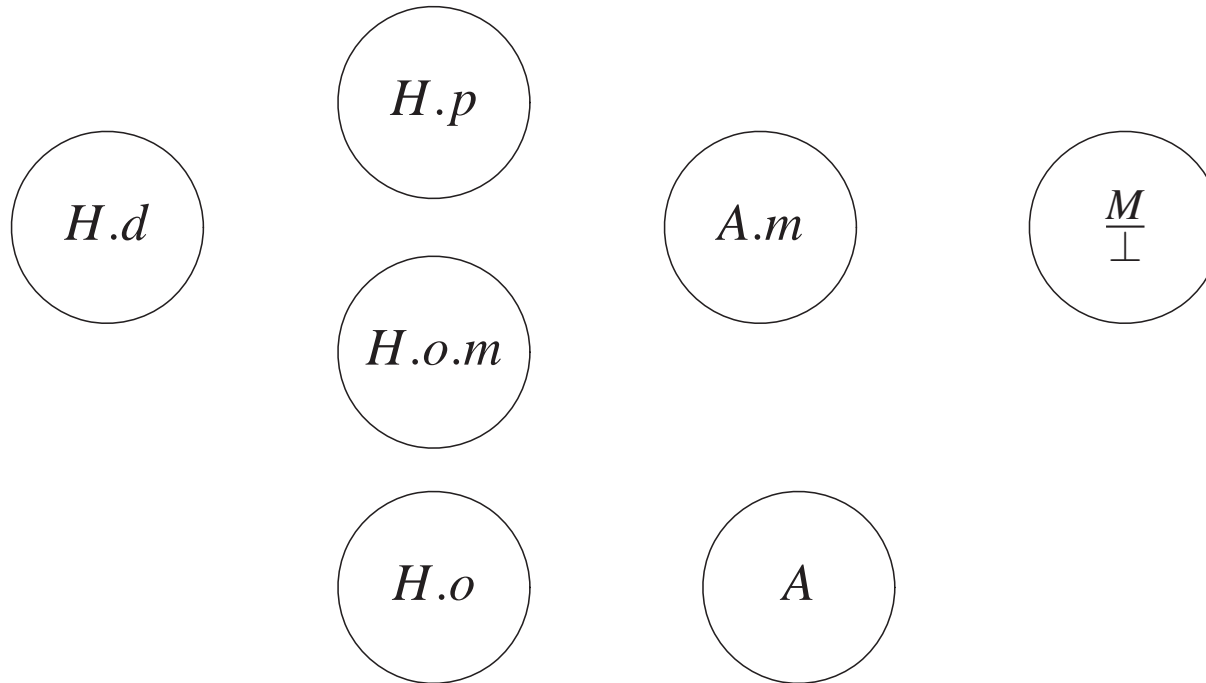
$$H.orgs \xleftarrow{10} AAA$$

$$H.preferred \xleftarrow{18} AAA.members$$

$$AAA.members \xleftarrow{4} M$$

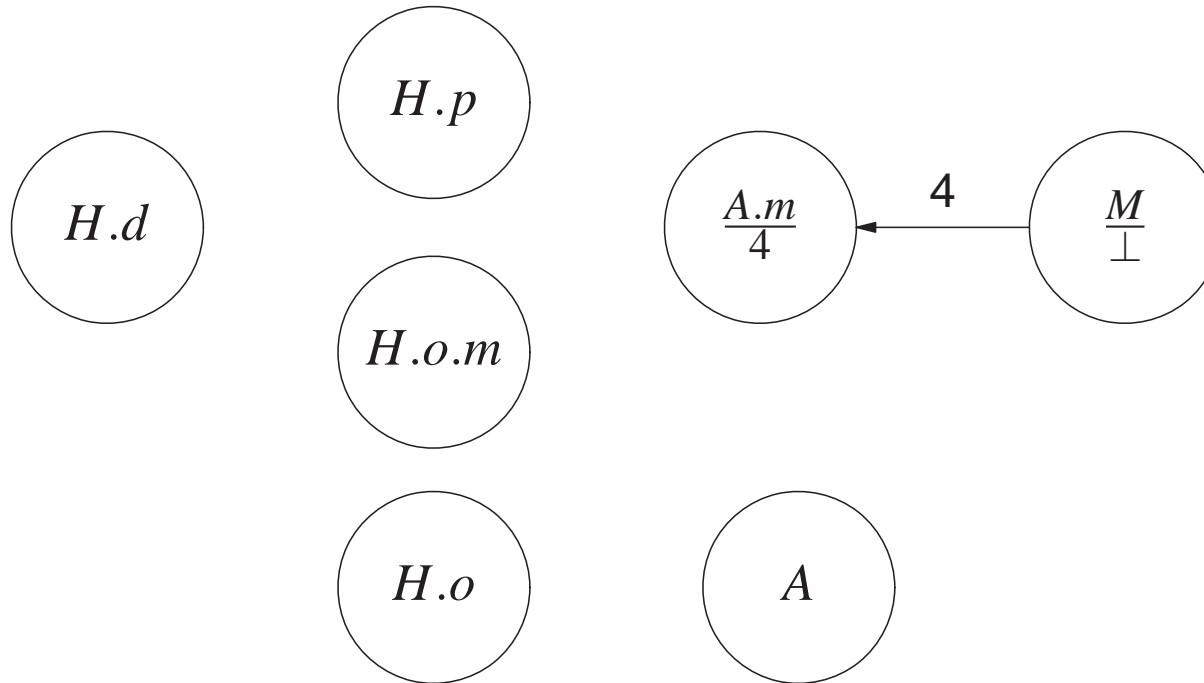
Suppose we've established a risk threshold of 20 for authorization.

Search Algorithm Example: 1



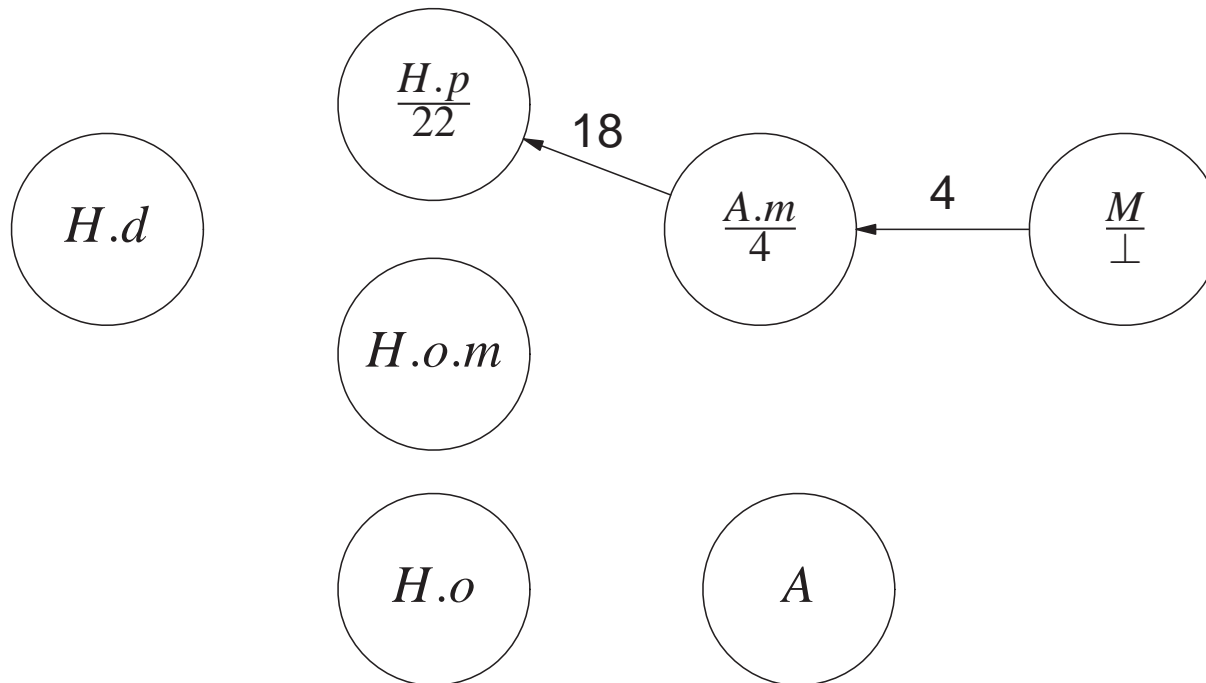
$$\kappa_M = 20$$

Search Algorithm Example: 2



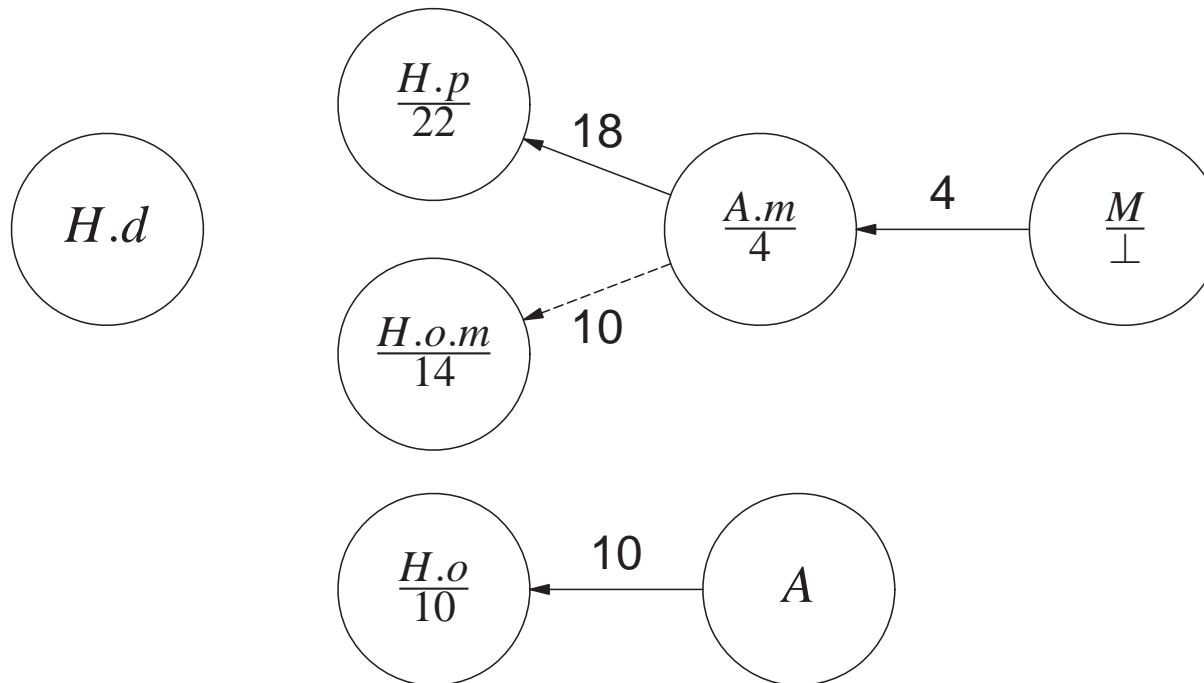
$$\kappa_M = 20$$

Search Algorithm Example: 3



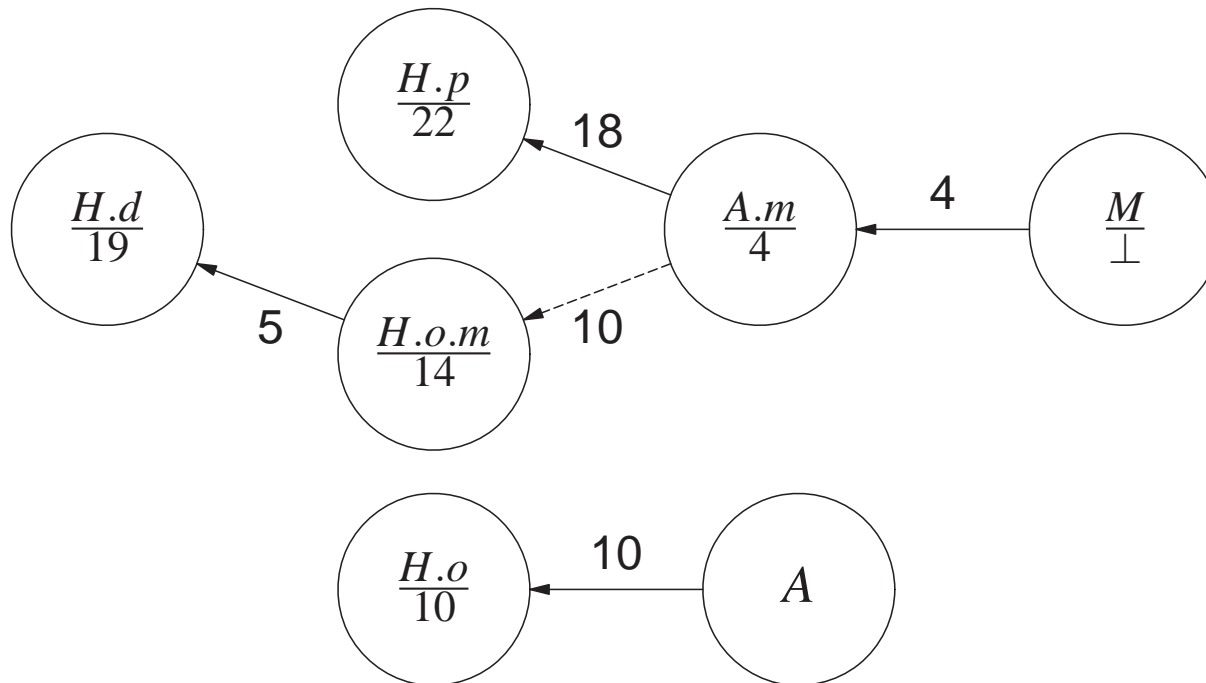
$$\kappa_M = 20$$

Search Algorithm Example: 4



$$\kappa_M = 20$$

Search Algorithm Example: 5



$$\kappa_M = 20$$

Horn Clause Specification of RT^R

```
ismem (role A R) B K <= credential A R (^ B) K.
```

```
ismem (role A R0) D K <=  
  credential A R0 (role B R1) K1,  
  ismem (role B R1) D K2,  
  aggregate K K1::K2::nil.
```

```
ismem (role A R0) E K <=  
  credential A R0 (linked_role B R1 R2) K1,  
  ismem (role B R1) D K2,  
  ismem (role D R2) E K3,  
  aggregate K K1::K2::K3::nil.
```

Conclusion

Take home message:

- TMSs are important technology for security in modern distributed systems.
- Programming logics provide good formal foundations, linguistic tools for TMSs.
- LolliMon provides right formula language and proof strategies for RT.
 - Conditional subgoals interleaves certificate retrieval and authorization.
 - Forward chaining proof strategy ensures termination.
 - Logical specification is scalable to full RT.

Future Work

Topics for future work:

- Implementation. System architecture, wire format, retrieval.
- Develop techniques for assigning risks to credentials.
- *Investigate other credential distribution/discovery schemes.*

<http://www.cs.uvm.edu/~skalka/skalka-pubs/skalka-projects.html>