

# Constructive Authorization Logics

Deepak Garg

Languages and Logics for Security

## Outline

- 1 Introduction
  - Goals and Motivation
  - Choosing a Logic
- 2 A Constructive Logic for Authorization
- 3 Proof Carrying Authorization
  - Enforcement with PCA
  - Issues and Limitations
- 4 Analysis of Authorization Policies
  - Non-interference
- 5 System Specification
  - Knowledge Modalities
  - Actual Modeling
  - Open Ends

## Outline

- 1 **Introduction**
  - Goals and Motivation
  - Choosing a Logic
- 2 A Constructive Logic for Authorization
- 3 Proof Carrying Authorization
  - Enforcement with PCA
  - Issues and Limitations
- 4 Analysis of Authorization Policies
  - Non-interference
- 5 System Specification
  - Knowledge Modalities
  - Actual Modeling
  - Open Ends

# What's an Authorization Logic?

- Must be a logic!
  - Proof theory, cut elimination
  - Semantics?
- Useful for Authorization
  - **Express** policies (“says”, delegation, ...)
  - **Enforce** policies (PCA, Bit-level implementations, ...)
  - **Explore** policies (Can my ex-employee access his files?)
- Emphasis on distributed scenarios

# What's an Authorization Logic?

- Must be a logic!
  - Proof theory, cut elimination
  - Semantics?
- Useful for Authorization
  - **Express** policies (“says”, delegation, ...)
  - **Enforce** policies (PCA, Bit-level implementations, ...)
  - **Explore** policies (Can my ex-employee access his files?)
- Emphasis on distributed scenarios

# What's an Authorization Logic?

- Must be a logic!
  - Proof theory, cut elimination
  - Semantics?
- Useful for Authorization
  - **Express** policies (“says”, delegation, ...)
  - **Enforce** policies (PCA, Bit-level implementations, ...)
  - **Explore** policies (Can my ex-employee access his files?)
- **Emphasis on distributed scenarios**

# Using the Logics

- Designed for
  - Proof carrying authorization
  - Specification of security systems
    - Also requires us to model states of knowledge
- **Not** Designed for
  - Authentication Properties (which key relates to which principal?)
  - Protocol specification

# Using the Logics

- Designed for
  - Proof carrying authorization
  - Specification of security systems
    - Also requires us to model states of knowledge
- **Not** Designed for
  - Authentication Properties (which key relates to which principal?)
  - Protocol specification

# Legacy (A little bit!)

- ABLP logic
- Binder [Soutei, Delegation Logic]
- Grey System at CyLab
- Proof Carrying Authentication [Appel, Felten]
  - Using lax logic

# Choosing a Logic

- “K says A” seems reasonable
  - Models statements from distinct principal
  - We write as  $\langle K \rangle A$ .
- “speaks for” is too general
  - $\langle K \rangle (K' \Rightarrow K)$ 
    - Does  $K$  know *all* predicates it delegated over?
- Constructive!
- Linearity is useful

# Choosing a Logic

- “K says A” seems reasonable
  - Models statements from distinct principal
  - We write as  $\langle K \rangle A$ .
- “speaks for” is too general
  - $\langle K \rangle (K' \Rightarrow K)$ 
    - Does  $K$  know *all* predicates it delegated over?
- Constructive!
- Linearity is useful

# Choosing a Logic

- “K says A” seems reasonable
  - Models statements from distinct principal
  - We write as  $\langle K \rangle A$ .
- “speaks for” is too general
  - $\langle K \rangle (K' \Rightarrow K)$ 
    - Does  $K$  know *all* predicates it delegated over?
- Constructive!
- Linearity is useful

# Choosing a Logic

- “K says A” seems reasonable
  - Models statements from distinct principal
  - We write as  $\langle K \rangle A$ .
- “speaks for” is too general
  - $\langle K \rangle (K' \Rightarrow K)$ 
    - Does  $K$  know *all* predicates it delegated over?
- Constructive!
- Linearity is useful

# Why Constructive Logic?

- In constructive logic,  $\vdash A \vee B$  implies  $\vdash A$  or  $\vdash B$
- Provides meaningful insights via proof normalization

# Why Constructive Logic?

$$\begin{aligned}
 p1 &: \text{all\_read}(F) \supset \text{mayread}(K, F) \\
 p2 &: \text{owner\_read}(F) \supset \text{owner}(K, F) \supset \text{mayread}(K, F) \\
 p3 &: \text{owner}(\text{Alice}, \text{foo.pdf}) \\
 M &= \lambda x : (\text{all\_read}(\text{foo.pdf}) \vee \text{owner\_read}(\text{foo.pdf})). \\
 &\quad \text{case } x \text{ of} \\
 &\quad \quad \text{inl } y \Rightarrow (p1) y \\
 &\quad \quad | \text{inr } z \Rightarrow (p2) z (p3)
 \end{aligned}$$

- Alice **constructs** a *closed* proof  
 $p : (\text{all\_read}(\text{foo.pdf}) \vee \text{owner\_read}(\text{foo.pdf}))$
- **Submits** proof  $(M p) : \text{mayread}(\text{Alice}, \text{foo.pdf})$
- **Gets** access
- Does  $(M p)$  tell us **why** Alice got access?

# Why Constructive Logic?

```

p1 :    all_read(F)  $\supset$  mayread(K, F)
p2 :    owner_read(F)  $\supset$  owner(K, F)  $\supset$  mayread(K, F)
p3 :    owner(Alice, foo.pdf)
M =     $\lambda x$  : (all_read(foo.pdf)  $\vee$  owner_read(foo.pdf)).
         case x of
           inl y  $\Rightarrow$  (p1) y
           | inr z  $\Rightarrow$  (p2) z (p3)
p :     (all_read(foo.pdf)  $\vee$  owner_read(foo.pdf))
(M p) : mayread(Alice, foo.pdf)

```

- Does (*M p*) tell us why Alice got access?
- In constructive logic, yes!
- *p* reduces to (inl *p'*) or (inr *p''*)
- (*M p*) reduces to ((*p*1) *p'*) or ((*p*2) (*p''*) (*p*3))

# Why Constructive Logic?

```

p1 :    all_read(F)  $\supset$  mayread(K, F)
p2 :    owner_read(F)  $\supset$  owner(K, F)  $\supset$  mayread(K, F)
p3 :    owner(Alice, foo.pdf)
M =     $\lambda x$  : (all_read(foo.pdf)  $\vee$  owner_read(foo.pdf)).
          case x of
            inl y  $\Rightarrow$  (p1) y
            | inr z  $\Rightarrow$  (p2) z (p3)
p :    (all_read(foo.pdf)  $\vee$  owner_read(foo.pdf))
(M p) : mayread(Alice, foo.pdf)

```

- Does not work classically.
- $p4 : \neg(\neg\text{all\_read}(F) \wedge \neg\text{owner\_read}(F))$
- Classically (using double negation elimination), this entails that  $(\text{all\_read}(\text{foo.pdf}) \vee \text{owner\_read}(\text{foo.pdf}))$
- Classical normal form is uninformative

# Need for Linearity

- Models “**use-once**” authorizations
- $(\langle Alice \rangle allow\_read(K, foo.pdf)) \multimap mayread(K, foo.pdf)$
- Alice can sign a **linear certificate** “ $allow\_read(Charlie, foo.pdf)$ ” to allow Charlie to read `foo.pdf` only once.
- Can model revocation
- $(\langle Alice \rangle revoke(K, foo.pdf)) \multimap (\langle Alice \rangle allow\_read(K, foo.pdf)) \multimap 1$
- Problem with revocation: What guarantees that this rule is used before file is read?

# Need for Linearity

- Models “use-once” authorizations
- $(\langle Alice \rangle allow\_read(K, foo.pdf)) \multimap mayread(K, foo.pdf)$
- Alice can sign a **linear certificate** “ $allow\_read(Charlie, foo.pdf)$ ” to allow Charlie to read `foo.pdf` only once.
- Can model revocation
- $(\langle Alice \rangle revoke(K, foo.pdf)) \multimap (\langle Alice \rangle allow\_read(K, foo.pdf)) \multimap 1$
- Problem with revocation: What guarantees that this rule is used before file is read?

# Need for Linearity

- Models “use-once” authorizations
- $(\langle Alice \rangle allow\_read(K, foo.pdf)) \multimap mayread(K, foo.pdf)$
- Alice can sign a **linear certificate** “ $allow\_read(Charlie, foo.pdf)$ ” to allow Charlie to read `foo.pdf` only once.
- Can model revocation
- $(\langle Alice \rangle revoke(K, foo.pdf)) \multimap (\langle Alice \rangle allow\_read(K, foo.pdf)) \multimap 1$
- Problem with revocation: What guarantees that this rule is used before file is read?

## Outline

- 1 Introduction
  - Goals and Motivation
  - Choosing a Logic
- 2 A Constructive Logic for Authorization**
- 3 Proof Carrying Authorization
  - Enforcement with PCA
  - Issues and Limitations
- 4 Analysis of Authorization Policies
  - Non-interference
- 5 System Specification
  - Knowledge Modalities
  - Actual Modeling
  - Open Ends

# Indexed Lax Logic

- $\langle K \rangle A$  is a **lax** modality
- Judgmentally presented
- Two categorical judgments
  - $A$  **true** (abbrev.  $A$ )
  - $K$  **affirms**  $A$  (cf.  $A$  **lax**)

$$\frac{\Gamma; \Delta \Rightarrow A}{\Gamma; \Delta \Rightarrow K \text{ affirms } A} \text{aff}$$

$$\frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A \text{ lax}}$$

$$\frac{\Gamma; \Delta \Rightarrow K \text{ affirms } A}{\Gamma; \Delta \Rightarrow \langle K \rangle A} \langle \rangle R$$

$$\frac{\Gamma \Rightarrow A \text{ lax}}{\Gamma \Rightarrow \bigcirc A}$$

$$\frac{\Gamma; \Delta, A \Rightarrow K \text{ affirms } C}{\Gamma; \Delta, \langle K \rangle A \Rightarrow K \text{ affirms } C} \langle \rangle L$$

$$\frac{\Gamma, A, \bigcirc A \Rightarrow C \text{ lax}}{\Gamma, \bigcirc A \Rightarrow C \text{ lax}}$$

# Indexed Lax Logic

- $\langle K \rangle A$  is a **lax** modality
- Judgmentally presented
- Two categorical judgments
  - $A$  **true** (abbrev.  $A$ )
  - $K$  **affirms**  $A$  (cf.  $A$  *lax*)

$$\frac{\Gamma; \Delta \Rightarrow A}{\Gamma; \Delta \Rightarrow K \text{ affirms } A} \text{aff}$$

$$\frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A \text{ lax}}$$

$$\frac{\Gamma; \Delta \Rightarrow K \text{ affirms } A}{\Gamma; \Delta \Rightarrow \langle K \rangle A} \langle \rangle R$$

$$\frac{\Gamma \Rightarrow A \text{ lax}}{\Gamma \Rightarrow \bigcirc A}$$

$$\frac{\Gamma; \Delta, A \Rightarrow K \text{ affirms } C}{\Gamma; \Delta, \langle K \rangle A \Rightarrow K \text{ affirms } C} \langle \rangle L$$

$$\frac{\Gamma, A, \bigcirc A \Rightarrow C \text{ lax}}{\Gamma, \bigcirc A \Rightarrow C \text{ lax}}$$

# Cut-Elimination

If  $\Gamma; \Delta_1 \Rightarrow A$   
and  $\Gamma; \Delta_2, A \Rightarrow \psi$      ( $\psi = C$  or  $K$  affirms  $C$ )  
then  $\Gamma; \Delta_1, \Delta_2 \Rightarrow \psi$

If  $\Gamma; \Delta_1 \Rightarrow K$  affirms  $A$   
and  $\Gamma; \Delta_2, A \Rightarrow K$  affirms  $C$   
then  $\Gamma; \Delta_1, \Delta_2 \Rightarrow K$  affirms  $C$

# Consequences of Cut-Elimination

- Consistency
  - $\not\vdash \perp$
  - $\not\vdash (\langle K \rangle \perp) \supset \perp$
- Sub-formula property (useful in policy analysis)
- Proof-normalization (very important)

## Outline

- 1 Introduction
  - Goals and Motivation
  - Choosing a Logic
- 2 A Constructive Logic for Authorization
- 3 Proof Carrying Authorization**
  - Enforcement with PCA
  - Issues and Limitations
- 4 Analysis of Authorization Policies
  - Non-interference
- 5 System Specification
  - Knowledge Modalities
  - Actual Modeling
  - Open Ends

# Recap of PCA

$p1 : \langle admin \rangle \text{mayread}(Alice, foo.pdf)$   
 $p2 : \langle admin \rangle (\langle Alice \rangle \text{allow\_read}(K, foo.pdf)$   
 $\quad \rightarrow \text{mayread}(K, foo.pdf))$

- 1 K  $\rightarrow$  Server: Let me read foo.pdf
- 2 Server  $\rightarrow$  K: Prove  $\langle admin \rangle \text{mayread}(K, foo.pdf)$
- 3 K assembles a proof M
- 4 K  $\rightarrow$  Server: M
- 5 Server verifies M (Grants or denies access)

# Recap of PCA

$p1 : \langle admin \rangle \text{mayread}(Alice, foo.pdf)$   
 $p2 : \langle admin \rangle (\langle Alice \rangle \text{allow\_read}(K, foo.pdf)$   
 $\quad \rightarrow \text{mayread}(K, foo.pdf))$

- Logic only models the authorization policy, not information flow

$\langle K \rangle \text{req\_read}(foo.pdf)$   
 $\rightarrow \text{hasproof}(K, \langle admin \rangle \text{mayread}(K, foo.pdf))$   
 $\rightarrow \text{allow\_access}(K, foo.pdf, read)$

- Concluding sequent always has form

$\langle K_1 \rangle A_1, \dots \langle K_n \rangle A_n \Rightarrow \langle K \rangle A$

- Assumptions discharged by signed certificates (X.509, ...)

# Recap of PCA

$p1 : \langle admin \rangle \text{mayread}(Alice, foo.pdf)$   
 $p2 : \langle admin \rangle (\langle Alice \rangle \text{allow\_read}(K, foo.pdf)$   
 $\quad \rightarrow \text{mayread}(K, foo.pdf))$

- Logic only models the authorization policy, not information flow

$\langle K \rangle \text{req\_read}(foo.pdf)$   
 $\rightarrow \text{hasproof}(K, \langle admin \rangle \text{mayread}(K, foo.pdf))$   
 $\rightarrow \text{allow\_access}(K, foo.pdf, read)$

- Concluding sequent always has form

$\langle K_1 \rangle A_1, \dots \langle K_n \rangle A_n \Rightarrow \langle K \rangle A$

- Assumptions discharged by signed certificates (X.509, ...)

# Enforcing Linearity

- *What's a linear certificate, and how do we make sure it is used once?*
- Centralized scenario:
  - Store all linear certificates at a trusted server
  - Each verifier contacts server when it verifies proof
  - Server marks certificates “used”
- Decentralized scenario:
  - Each linear certificate mentions a ratifier (decided by signer)
  - Ratifiers contacted at time of proof verification
  - Ratifiers **atomically** mark certificates “used”
  - Very inefficient

# Enforcing Linearity

- *What's a linear certificate, and how do we make sure it is used once?*
- Centralized scenario:
  - Store all linear certificates at a trusted server
  - Each verifier contacts server when it verifies proof
  - Server marks certificates “used”
- Decentralized scenario:
  - Each linear certificate mentions a ratifier (decided by signer)
  - Ratifiers contacted at time of proof verification
  - Ratifiers **atomically** mark certificates “used”
  - Very inefficient

# Enforcing Linearity

- *What's a linear certificate, and how do we make sure it is used once?*
- Centralized scenario:
  - Store all linear certificates at a trusted server
  - Each verifier contacts server when it verifies proof
  - Server marks certificates “used”
- Decentralized scenario:
  - Each linear certificate mentions a ratifier (decided by signer)
  - Ratifiers contacted at time of proof verification
  - Ratifiers **atomically** mark certificates “used”
  - Very inefficient

# Namspaces and Key Infrastructure

- Central naming conventions
  - When “Charlie” signs `mayread(Alice, foo.pdf)`, who does he call “Alice” and “mayread”?
- Severly limits the possibility of user defined predicates or requires namespace conventions
- Infrastructure for managing keys is needed (PGP, SPKI, ...)
- Mapping keys to principals (“speaks for” is ideal here, but we abandoned it!)

# Namspaces and Key Infrastructure

- Central naming conventions
  - When “Charlie” signs `mayread(Alice, foo.pdf)`, who does he call “Alice” and “mayread”?
- Severly limits the possibility of user defined predicates or requires namespace conventions
- Infrastructure for managing keys is needed (PGP, SPKI, ...)
- Mapping keys to principals (“speaks for” is ideal here, but we abandoned it!)

# Limitation of Lax Logic: The Shop Assistant Problem

- A shop assistant ( $A$ ) wants to sell an item at price  $P$
- He has no authority to determine if  $P$  is a fair price, so he delegates the decision to the owner ( $O$ ).

$p1 : \langle A \rangle (\langle O \rangle \text{sell}(P) \supset \text{sell}(P))$

- Owner allows the sale if the price is fair

$p2 : \langle O \rangle (\text{fair}(P) \supset \text{sell}(P))$

- Assistant **but not the owner** believes that  $\text{fair}(P)$ .

$p3 : \langle A \rangle \text{fair}(P)$

Should the assistant be allowed to sell the item?

Is  $\langle A \rangle \text{sell}(P)$  provable from  $p1, p2, p3$ ?

# Limitation of Lax Logic: The Shop Assistant Problem

- A shop assistant ( $A$ ) wants to sell an item at price  $P$
- He has no authority to determine if  $P$  is a fair price, so he delegates the decision to the owner ( $O$ ).

$$p1 : \langle A \rangle (\langle O \rangle \text{sell}(P) \supset \text{sell}(P))$$

- Owner allows the sale if the price is fair

$$p2 : \langle O \rangle (\text{fair}(P) \supset \text{sell}(P))$$

- Assistant **but not the owner** believes that  $\text{fair}(P)$ .

$$p3 : \langle A \rangle \text{fair}(P)$$

Should the assistant be allowed to sell the item?

Is  $\langle A \rangle \text{sell}(P)$  provable from  $p1, p2, p3$ ?

# Limitation of Law Logic: The Shop Assistant Problem

- A shop assistant (A) wants to sell an item at price P
- He has no authority to determine if P is a fair price, so he delegates the decision to the owner (O).

$$p1 : \langle A \rangle (\langle O \rangle \text{sell}(P) \supset \text{sell}(P))$$

- Owner allows the sale if the price is fair

$$p2 : \langle O \rangle (\text{fair}(P) \supset \text{sell}(P))$$

- Assistant **but not the owner** believes that  $\text{fair}(P)$ .

$$p3 : \langle A \rangle \text{fair}(P)$$

Should the assistant be allowed to sell the item?

Is  $\langle A \rangle \text{sell}(P)$  provable from  $p1, p2, p3$ ?

# Limitation of Lax Logic: The Shop Assistant Problem

$p1: \langle A \rangle (\langle O \rangle \text{sell}(P) \supset \text{sell}(P))$

$p2: \langle O \rangle (\text{fair}(P) \supset \text{sell}(P))$

$p3: \langle A \rangle \text{fair}(P)$

Goal:  $\langle A \rangle \text{sell}(P)$

- Intuitively, goal should not be provable:  $O$  does not believe fairness of  $P$
- But, if  $\langle K \rangle$  is lax, then goal is provable!
- Completely counterintuitive
- Problematic axiom is “unit” law:  $\vdash A \supset \langle K \rangle A$
- Can use weaker logics (ongoing work)

# Limitation of Lax Logic: The Shop Assistant Problem

$p1: \langle A \rangle (\langle O \rangle \text{sell}(P) \supset \text{sell}(P))$

$p2: \langle O \rangle (\text{fair}(P) \supset \text{sell}(P))$

$p3: \langle A \rangle \text{fair}(P)$

Goal:  $\langle A \rangle \text{sell}(P)$

- Intuitively, goal should not be provable:  $O$  does not believe fairness of  $P$
- But, if  $\langle K \rangle$  is lax, then goal is provable!
- Completely counterintuitive
  
- Problematic axiom is “unit” law:  $\vdash A \supset \langle K \rangle A$
- Can use weaker logics (ongoing work)

# Limitation of Lax Logic: The Shop Assistant Problem

$p1 : \langle A \rangle (\langle O \rangle \text{sell}(P) \supset \text{sell}(P))$

$p2 : \langle O \rangle (\text{fair}(P) \supset \text{sell}(P))$

$p3 : \langle A \rangle \text{fair}(P)$

Goal:  $\langle A \rangle \text{sell}(P)$

- Intuitively, goal should not be provable:  $O$  does not believe fairness of  $P$
- But, if  $\langle K \rangle$  is lax, then goal is provable!
- Completely counterintuitive
  
- Problematic axiom is “unit” law:  $\vdash A \supset \langle K \rangle A$
- Can use weaker logics (ongoing work)

## Outline

- 1 Introduction
  - Goals and Motivation
  - Choosing a Logic
- 2 A Constructive Logic for Authorization
- 3 Proof Carrying Authorization
  - Enforcement with PCA
  - Issues and Limitations
- 4 Analysis of Authorization Policies**
  - Non-interference**
- 5 System Specification
  - Knowledge Modalities
  - Actual Modeling
  - Open Ends

# Analysis of Authorization Policies

- How do we know that a policy is correct?

- Simple question:

“Does the policy allow Alice to read foo.pdf?”

- Can be answered with a theorem prover for the logic

- More complicated question:

“If Alice signs something of the form  $\text{mayread}(K, F)$ , will it affect any decision drawn from the policy?”

(Does Alice have any control over predicate  $\text{mayread}$ )

- Cannot be answered with a theorem prover
- Needs a meta-level theorem about the policy.

# Analysis of Authorization Policies

- How do we know that a policy is correct?
- Simple question:  
“Does the policy allow Alice to read foo.pdf?”
  - Can be answered with a theorem prover for the logic
- More complicated question:  
“If Alice signs something of the form  $\text{mayread}(K, F)$ , will it affect any decision drawn from the policy?”  
(Does Alice have any control over predicate  $\text{mayread}$ )
  - Cannot be answered with a theorem prover
  - Needs a meta-level theorem about the policy.

# Analysis of Authorization Policies

- How do we know that a policy is correct?
- Simple question:  
“Does the policy allow Alice to read foo.pdf?”
  - Can be answered with a theorem prover for the logic
- More complicated question:  
“If Alice signs something of the form  $\text{mayread}(K, F)$ , will it affect any decision drawn from the policy?”  
(Does Alice have any control over predicate  $\text{mayread}$ )
  - Cannot be answered with a theorem prover
  - Needs a meta-level theorem about the policy.

# Analysis of Authorization Policies

- How do we know that a policy is correct?
- Simple question:  
“Does the policy allow Alice to read foo.pdf?”
  - Can be answered with a theorem prover for the logic
- More complicated question:  
“If Alice signs something of the form  $\text{mayread}(K, F)$ , will it affect any decision drawn from the policy?”  
(Does Alice have any control over predicate  $\text{mayread}$ )
  - Cannot be answered with a theorem prover
  - Needs a meta-level theorem about the policy.

# Non-interference Property

- Property of the following form:

If  $\Gamma, A \Rightarrow \psi$  and

<Some VERIFIABLE condition on  $\Gamma, A, \psi$ >

then  $\Gamma \Rightarrow \psi$

- Assumption  $A$  does not “interfere” with policy  $\Gamma$  for conclusions  $\psi$
- Can be formulated very generally for large classes of  $\Gamma, A$  and  $\psi$

# Non-interference Property

- Property of the following form:

If  $\Gamma, A \Rightarrow \psi$  and

$\langle$ Some VERIFIABLE condition on  $\Gamma, A, \psi$  $\rangle$

then  $\Gamma \Rightarrow \psi$

- Assumption  $A$  does not “interfere” with policy  $\Gamma$  for conclusions  $\psi$
- Can be formulated very generally for large classes of  $\Gamma, A$  and  $\psi$

# Simple Non-interference

If  $\Gamma, \langle K \rangle A \Rightarrow \psi$  and

$\langle K \notin \Gamma, \psi \text{ and } \Gamma, \psi \text{ are quantifier-free} \rangle$

then  $\Gamma \Rightarrow \psi$

- Can be used to show that irrelevant principals cannot interfere!
- Quite weak for practical purposes
  - Most realistic policies have quantifiers

# Simple Non-interference

If  $\Gamma, \langle K \rangle A \Rightarrow \psi$  and

$\langle K \notin \Gamma, \psi \text{ and } \Gamma, \psi \text{ are quantifier-free} \rangle$

then  $\Gamma \Rightarrow \psi$

- Can be used to show that irrelevant principals cannot interfere!
- Quite weak for practical purposes
  - Most realistic policies have quantifiers

# Non-interference using affirmation flow

$p1 : \langle admin \rangle \text{mayread}(Alice, foo.pdf)$   
 $p2 : \langle admin \rangle (\langle Alice \rangle \text{allow\_read}(K, foo.pdf) \multimap \text{mayread}(K, foo.pdf))$

- Using the policy, derive a “flow relation”:

$admin.((Alice.\text{allow\_read}) \leq \text{mayread})$

“Inside the admin’s domain, a statement by Alice regarding `allow_read` may affect the truth of a statement containing `mayread`”

- $\leq$  is reflexive, transitive + other properties (related to lax modality)
- Depends on the policy

# Non-interference using affirmation flow

$$\begin{aligned} p1 &: \langle admin \rangle \text{mayread}(Alice, foo.pdf) \\ p2 &: \langle admin \rangle (\langle Alice \rangle \text{allow\_read}(K, foo.pdf) \\ &\quad \rightarrow \text{mayread}(K, foo.pdf)) \end{aligned}$$

- Using the policy, derive a “flow relation”:  
 $admin.((Alice.allow\_read) \leq \text{mayread})$   
“Inside the admin’s domain, a statement by Alice regarding `allow_read` may affect the truth of a statement containing `mayread`”
- $\leq$  is reflexive, transitive + other properties (related to lax modality)
- Depends on the policy

# Non-interference using affirmation flow

$p1 : \langle \text{admin} \rangle \text{mayread}(\text{Alice}, \text{foo.pdf})$   
 $p2 : \langle \text{admin} \rangle (\langle \text{Alice} \rangle \text{allow\_read}(K, \text{foo.pdf})$   
 $\quad \quad \quad \rightarrow \text{mayread}(K, \text{foo.pdf}))$

If  $\Gamma, A \Rightarrow \psi$  and

<Nothing in  $A$  is below anything in  $\psi$  in the relation  $\leq$  derived from  $\Gamma, A, \psi$ >

then  $\Gamma \Rightarrow \psi$

- $\leq$  can be constructed and verified automatically

# Non-interference using affirmation flow

$p1 : \langle admin \rangle \text{mayread}(Alice, foo.pdf)$   
 $p2 : \langle admin \rangle (\langle Alice \rangle \text{allow\_read}(K, foo.pdf) \multimap \text{mayread}(K, foo.pdf))$

- $Alice.\text{mayread} \not\leq admin.\text{mayread}$
- $p1, p2, \langle Alice \rangle \text{mayread}(K, F) \Rightarrow \langle admin \rangle \text{mayread}(K', F')$   
if and only if  
 $p1, p2 \Rightarrow \langle admin \rangle \text{mayread}(K', F')$
- Alice has no jurisdiction on predicate `mayread`!  
(Makes intuitive sense: Alice's jurisdiction is over the predicate `allow_read`)

# Non-interference using affirmation flow

$p1 : \langle admin \rangle \text{mayread}(Alice, foo.pdf)$   
 $p2 : \langle admin \rangle (\langle Alice \rangle \text{allow\_read}(K, foo.pdf) \multimap \text{mayread}(K, foo.pdf))$

- $Alice.\text{mayread} \not\leq admin.\text{mayread}$
- $p1, p2, \langle Alice \rangle \text{mayread}(K, F) \Rightarrow \langle admin \rangle \text{mayread}(K', F')$   
if and only if  
 $p1, p2 \Rightarrow \langle admin \rangle \text{mayread}(K', F')$
- Alice has no jurisdiction on predicate `mayread`!  
(Makes intuitive sense: Alice's jurisdiction is over the predicate `allow_read`)

## Outline

- 1 Introduction
  - Goals and Motivation
  - Choosing a Logic
- 2 A Constructive Logic for Authorization
- 3 Proof Carrying Authorization
  - Enforcement with PCA
  - Issues and Limitations
- 4 Analysis of Authorization Policies
  - Non-interference
- 5 System Specification**
  - Knowledge Modalities
  - Actual Modeling
  - Open Ends

# Specifying Secure Systems in Logic

- Given an *implemented* system with
  - Rules for transferring and updating **data/information/objects**
  - **Authorizations** for allowing transfers
- Example of rule:  
If “Server has information” and “K may access information”, then “K has information”
- Objective is to model **both** information and authorizations in one logic
- How do we model “K has information”?
  - Model “information” as a logical formula (usually a predicate)
  - Add modalities for modelling “K has”

# Specifying Secure Systems in Logic

- Given an *implemented* system with
  - Rules for transferring and updating **data/information/objects**
  - **Authorizations** for allowing transfers
- Example of rule:  
If “Server has information” and “K may access information”, then “K has information”
- Objective is to model **both** information and authorizations in one logic
- How do we model “K has information”?
  - Model “information” as a logical formula (usually a predicate)
  - Add modalities for modelling “K has”

# Knowledge Modalities: $\llbracket K \rrbracket$ and $[K]$

- $\llbracket K \rrbracket$ A reads “K knows that A is true”
- Models *facts*
  - $\llbracket K \rrbracket$ round(*world*)
  - $\llbracket K \rrbracket$ contents(*F*, *C*)
  - $\llbracket K \rrbracket$ password(“*abc*”)
- $[K]$ A reads “K possesses A ”
- Models objects which should not be replicated (linear)
  - $[K]$ cash(\$10)
  - $[K]$ check(*K'*, \$1000)
  - $[K]$ lock(*F*)

# Formal description of $\llbracket K \rrbracket$ and $[K]$

- $\llbracket K \rrbracket$  and  $[K]$  are intuitionistic S4  $\Box$  modalities
- Two new categorical judgments: *K knows A* and *K has A*
- Occur only on left (like *A valid* in S4)

$$\frac{\Gamma |_{K}; \Delta |_{K} \Rightarrow A}{\Gamma; \Delta |_{K} \Rightarrow [K]A} \Box R$$

$$\frac{\Gamma |_{K}; \cdot \Rightarrow A}{\Gamma; \cdot \Rightarrow \llbracket K \rrbracket A} \Box R$$

$$\frac{\Gamma; \Delta, K \text{ has } A \Rightarrow \psi}{\Gamma; \Delta, [K]A \Rightarrow \psi} \Box L$$

$$\frac{\Gamma, K \text{ knows } A; \Delta \Rightarrow \psi}{\Gamma; \Delta, \llbracket K \rrbracket A \Rightarrow \psi} \Box L$$

$$\frac{\Gamma; \Delta, A \Rightarrow \psi}{\Gamma; \Delta, K \text{ has } A \Rightarrow \psi} \text{has}$$

$$\frac{\Gamma, K \text{ knows } A; \Delta, A \Rightarrow \psi}{\Gamma, K \text{ knows } A; \Delta \Rightarrow \psi} \text{knows}$$

# Formal description of $\llbracket K \rrbracket$ and $[K]$

- $\llbracket K \rrbracket$  and  $[K]$  are intuitionistic S4  $\Box$  modalities
- Two new categorical judgments: *K knows A* and *K has A*
- Occur only on left (like *A valid* in S4)

$$\frac{\Gamma |_{K}; \Delta |_{K} \Rightarrow A}{\Gamma; \Delta |_{K} \Rightarrow [K]A} \Box R$$

$$\frac{\Gamma |_{K}; \cdot \Rightarrow A}{\Gamma; \cdot \Rightarrow \llbracket K \rrbracket A} \Box\Box R$$

$$\frac{\Gamma; \Delta, K \text{ has } A \Rightarrow \psi}{\Gamma; \Delta, [K]A \Rightarrow \psi} \Box L$$

$$\frac{\Gamma, K \text{ knows } A; \Delta \Rightarrow \psi}{\Gamma; \Delta, \llbracket K \rrbracket A \Rightarrow \psi} \Box\Box L$$

$$\frac{\Gamma; \Delta, A \Rightarrow \psi}{\Gamma; \Delta, K \text{ has } A \Rightarrow \psi} \text{has}$$

$$\frac{\Gamma, K \text{ knows } A; \Delta, A \Rightarrow \psi}{\Gamma, K \text{ knows } A; \Delta \Rightarrow \psi} \text{knows}$$

# Cut-elimination

If  $\Gamma|_K; \cdot \Rightarrow A$

and  $\Gamma, K \text{ knows } A; \Delta \Rightarrow \psi$       ( $\psi = C$  or  $K$  affirms  $C$ )

then  $\Gamma; \Delta \Rightarrow \psi$

If  $\Gamma|_K; \Delta_1|_K \Rightarrow A$

and  $\Gamma; \Delta_2, K \text{ has } A \Rightarrow \psi$

then  $\Gamma; \Delta_1|_K, \Delta_2 \Rightarrow \psi$

# Actual Modeling

$$\begin{aligned} & \llbracket \text{Server} \rrbracket \text{contents}(F, C) \supset \\ & \quad \langle \text{admin} \rangle \text{mayread}(K, F) \supset \\ & \quad \llbracket K \rrbracket \text{contents}(F, C) \end{aligned}$$

- Knowledge Semantics

$$\llbracket K \rrbracket \text{contents}(F, C) \longrightarrow ? \text{ (Store)}$$

- Authorization Semantics

$$\langle \text{admin} \rangle \text{mayread}(K, F) \longrightarrow ? \text{ (ACL, Certificate or Proof)}$$

- Ensure that implementation conforms to specification
  - No requirement of completeness
- Last lecture's work a step in this direction
- A lot more needs to be done

# Actual Modeling

$$\begin{aligned} & \llbracket \text{Server} \rrbracket \text{contents}(F, C) \supset \\ & \quad \langle \text{admin} \rangle \text{mayread}(K, F) \supset \\ & \quad \quad \llbracket K \rrbracket \text{contents}(F, C) \end{aligned}$$

- Knowledge Semantics

$$\llbracket K \rrbracket \text{contents}(F, C) \longrightarrow ? \text{ (Store)}$$

- Authorization Semantics

$$\langle \text{admin} \rangle \text{mayread}(K, F) \longrightarrow ? \text{ (ACL, Certificate or Proof)}$$

- Ensure that implementation conforms to specification
  - No requirement of completeness
- Last lecture's work a step in this direction
- A lot more needs to be done

# Actual Modeling

$$\begin{aligned} & \llbracket \text{Server} \rrbracket \text{contents}(F, C) \supset \\ & \quad \langle \text{admin} \rangle \text{mayread}(K, F) \supset \\ & \quad \quad \llbracket K \rrbracket \text{contents}(F, C) \end{aligned}$$

- Knowledge Semantics

$$\llbracket K \rrbracket \text{contents}(F, C) \longrightarrow ? \text{ (Store)}$$

- Authorization Semantics

$$\langle \text{admin} \rangle \text{mayread}(K, F) \longrightarrow ? \text{ (ACL, Certificate or Proof)}$$

- Ensure that implementation conforms to specification

- No requirement of completeness
- Last lecture's work a step in this direction
- A lot more needs to be done

# Actual Modeling

$$\begin{aligned} & \llbracket \text{Server} \rrbracket \text{contents}(F, C) \supset \\ & \quad \langle \text{admin} \rangle \text{mayread}(K, F) \supset \\ & \quad \quad \llbracket K \rrbracket \text{contents}(F, C) \end{aligned}$$

- Knowledge Semantics

$$\llbracket K \rrbracket \text{contents}(F, C) \longrightarrow ? \text{ (Store)}$$

- Authorization Semantics

$$\langle \text{admin} \rangle \text{mayread}(K, F) \longrightarrow ? \text{ (ACL, Certificate or Proof)}$$

- Ensure that implementation conforms to specification

- No requirement of completeness

- Last lecture's work a step in this direction

- A lot more needs to be done

# Why do all this?

- Prove properties of the specification
- Since implementation conforms, it obeys the same properties
- Banking system: account books remain balanced
- University registration: credit limits respected
- Secure file system: every access preceded by authorization

# Open End: Time in Certificates

- Almost all certificate schemes allow expiration time
  - How do we model this?
- 
- Add a new modality for time:  $A@I$ ,  $I$  is an interval of time
  - Certificates become:  $\langle K \rangle(A@I)$
  - Logic is Hybrid
  - Problem: how do we enforce  $A@I$  in general?
- 
- Tie time to says modality:  $\langle K, I \rangle A$
  - Closely tied to PCA implementations
  - Question: What rules do we choose for  $\langle K, I \rangle A$ ?
  - Will lax work?

# Open End: Time in Certificates

- Almost all certificate schemes allow expiration time
- How do we model this?
  
- Add a new modality for time:  $A@I$ ,  $I$  is an interval of time
- Certificates become:  $\langle K \rangle(A@I)$
- Logic is Hybrid
- Problem: how do we enforce  $A@I$  in general?
  
- Tie time to says modality:  $\langle K, I \rangle A$
- Closely tied to PCA implementations
- Question: What rules do we choose for  $\langle K, I \rangle A$ ?
- Will lax work?

# Open End: Time in Certificates

- Almost all certificate schemes allow expiration time
- How do we model this?
  
- Add a new modality for time:  $A@I$ ,  $I$  is an interval of time
- Certificates become:  $\langle K \rangle(A@I)$
- Logic is Hybrid
- Problem: how do we enforce  $A@I$  in general?
  
- Tie time to says modality:  $\langle K, I \rangle A$
- Closely tied to PCA implementations
- Question: What rules do we choose for  $\langle K, I \rangle A$ ?
- Will lax work?

# Open End: Complexity Questions

(Based on Joint Work with Martín Abadi)

- Theorem provers and proof-search inevitable for policy analysis
- How hard are they?
  
- Answer via translation to classical S4
- $\lceil \langle K \rangle A \rceil = \Box(K \vee \lceil A \rceil)$
- Sound and complete
  
- Consequences:
  - 1 Propositional fragment decidable (PSPACE)
  - 2 Complete Kripke semantics (don't seem very useful, though)
  - 3 Finite model property

# Open End: Complexity Questions

(Based on Joint Work with Martín Abadi)

- Theorem provers and proof-search inevitable for policy analysis
- How hard are they?
  
- Answer via translation to classical S4
- $\lceil \langle K \rangle A \rceil = \Box(K \vee \lceil A \rceil)$
- Sound and complete
  
- Consequences:
  - 1 Propositional fragment decidable (PSPACE)
  - 2 Complete Kripke semantics (don't seem very useful, though)
  - 3 Finite model property

# Open End: Complexity Questions

(Based on Joint Work with Martín Abadi)

- Theorem provers and proof-search inevitable for policy analysis
- How hard are they?
  
- Answer via translation to classical S4
- $\lceil \langle K \rangle A \rceil = \Box(K \vee \lceil A \rceil)$
- Sound and complete
  
- Consequences:
  - 1 Propositional fragment decidable (PSPACE)
  - 2 Complete Kripke semantics (don't seem very useful, though)
  - 3 Finite model property