

Proof Theory for Authorization Logic and its Application to a Practical File System

Deepak Garg

Thesis Oral

Committee

Frank Pfenning (Chair)

Martín Abadi (UCSC & MSR-SVC)

Lujo Bauer

Anupam Datta

Robert Harper

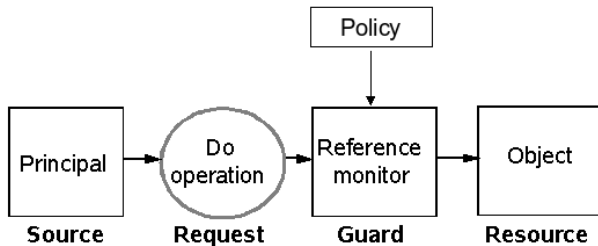
Outline

- 1 Background
- 2 Overview of Thesis: Motivation and Contributions
- 3 BL_S : A Logic for Static Authorizations
- 4 BL: A Logic for Dynamic Authorizations
- 5 PCFS: The File System
- 6 Conclusion

Outline

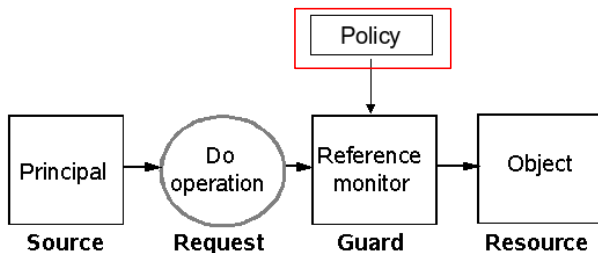
- 1 Background
- 2 Overview of Thesis: Motivation and Contributions
- 3 BL_S : A Logic for Static Authorizations
- 4 BL: A Logic for Dynamic Authorizations
- 5 PCFS: The File System
- 6 Conclusion

The Traditional Architecture of Access Control



Source: Lampson et al. '92

The Traditional Architecture of Access Control



Source: Lampson et al. '92

Policy is the main concern of this thesis

Relevant Questions

- How is the policy **represented**?
- How is the policy **enforced** at the guard?

Background: Policy Representation

- Represent policy as logical formulas [LABW'92,ABLP'93]
- Why logic?
 - ▶ High-level representation; coincides with natural understanding
 - ▶ Unambiguous interpretation
 - ▶ Accountability (keep track of who is accountable for access)
 - ▶ Easy enforcement
- Which logic? Special logics called *authorization logics*
 - ▶ Many authorization logics exist
 - ▶ Many logic-based declarative policy languages also exist

Background: Policy Representation

- Represent policy as logical formulas [LABW'92,ABLP'93]
- Why logic?
 - ▶ High-level representation; coincides with natural understanding
 - ▶ Unambiguous interpretation
 - ▶ Accountability (keep track of who is accountable for access)
 - ▶ Easy enforcement
- Which logic? Special logics called **authorization logics**
 - ▶ Many authorization logics exist
 - ▶ Many logic-based declarative policy languages also exist

Background: Policy Representation

- Represent policy as logical formulas [LABW'92,ABLP'93]
- Why logic?
 - ▶ High-level representation; coincides with natural understanding
 - ▶ Unambiguous interpretation
 - ▶ Accountability (keep track of who is accountable for access)
 - ▶ Easy enforcement
- Which logic? Special logics called **authorization logics**
 - ▶ Many authorization logics exist
 - ▶ Many logic-based declarative policy languages also exist

Background: Policy Representation in Logic

Key Ideas

- Basic facts become atomic formulas
- Example: “Bob should be allowed to read foo.txt”
- Represented as `may(Bob, foo.txt, read)`

Background: Policy Representation in Logic

Key Ideas

- What if several principals may contribute to an authorization policy?
- If principal k makes statement s , represent as formula (k says s)
[LABW'92,ABLP'93]
- Example: Alice states “Bob should be allowed to read foo.txt”
- Represented as $\text{Alice says } \text{may}(\text{Bob}, \text{foo.txt}, \text{read})$
- Statements are concretely established via **digitally signed certificates**
- In the above example, Alice may sign the formula $\text{may}(\text{Bob}, \text{foo.txt}, \text{read})$ with her **private key**

We will return to formulas k says s later

Background: Policy Representation in Logic

Key Ideas

- Implication represents cause-and-effect, as usual
- Example: “If Alice says Bob should be allowed to read foo.txt, then Bob should be allowed to read foo.txt”
- Represented as

admin says ((Alice says $\text{may}(\text{Bob}, \text{foo.txt}, \text{read})$)
 $\supset \text{may}(\text{Bob}, \text{foo.txt}, \text{read})$)

Background: From Representation to Enforcement

- Summary of policy representation in logic
 - ▶ Principals sign statements; represent as k says s
 - ▶ Statements together constitute the policy
$$\Gamma = \{k_1 \text{ says } s_1, \dots, k_n \text{ says } s_n\}$$
- How is this representation used for **enforcement** in the guard?
- Idea! Use **logical inference** for enforcement
- Allow access only if Γ entails a fixed formula s , i.e., $\Gamma \vdash s$
 - ▶ s may have the form $(\text{admin says } \text{may}(k, f, \text{read}))$
- Completely rigorous, formal
- Question: Isn't logical inference undecidable?
- Answer: Yes, so require that each *access request be accompanied by a proof of $\Gamma \vdash s$* .
 - ▶ The guard must only **verify the proof** – simple, linear.
- This is called proof-carrying authorization (PCA) [AF'99,Bau'03]

Background: From Representation to Enforcement

- Summary of policy representation in logic
 - ▶ Principals sign statements; represent as k says s
 - ▶ Statements together constitute the policy
$$\Gamma = \{k_1 \text{ says } s_1, \dots, k_n \text{ says } s_n\}$$
- How is this representation used for **enforcement** in the guard?
- **Idea!** Use **logical inference** for enforcement
- Allow access only if Γ entails a fixed formula s , i.e., $\Gamma \vdash s$
 - ▶ s may have the form $(\text{admin says}_{\text{may}}(k, f, \text{read}))$
- Completely rigorous, formal
- Question: Isn't logical inference undecidable?
- Answer: Yes, so require that each *access request be accompanied by a proof of $\Gamma \vdash s$* .
 - ▶ The guard must only **verify the proof** – simple, linear.
- This is called proof-carrying authorization (PCA) [AF'99,Bau'03]

Background: From Representation to Enforcement

- Summary of policy representation in logic
 - ▶ Principals sign statements; represent as k says s
 - ▶ Statements together constitute the policy
$$\Gamma = \{k_1 \text{ says } s_1, \dots, k_n \text{ says } s_n\}$$
- How is this representation used for **enforcement** in the guard?
- **Idea!** Use **logical inference** for enforcement
- Allow access only if Γ entails a fixed formula s , i.e., $\Gamma \vdash s$
 - ▶ s may have the form $(\text{admin says}_{\text{may}}(k, f, \text{read}))$
- Completely rigorous, formal
- Question: Isn't logical inference undecidable?
- Answer: Yes, so require that each *access request be accompanied by a proof of $\Gamma \vdash s$* .
 - ▶ The guard must only **verify the proof** – simple, linear.
- This is called proof-carrying authorization (PCA) [AF'99,Bau'03]

Background: From Representation to Enforcement

- Summary of policy representation in logic
 - ▶ Principals sign statements; represent as k says s
 - ▶ Statements together constitute the policy
$$\Gamma = \{k_1 \text{ says } s_1, \dots, k_n \text{ says } s_n\}$$
- How is this representation used for **enforcement** in the guard?
- Idea! Use **logical inference** for enforcement
- Allow access only if Γ entails a fixed formula s , i.e., $\Gamma \vdash s$
 - ▶ s may have the form $(\text{admin says } \text{may}(k, f, \text{read}))$
- Completely rigorous, formal
- Question: Isn't logical inference undecidable?
- Answer: Yes, so require that each *access request be accompanied by a proof of $\Gamma \vdash s$* .
 - ▶ The guard must only **verify the proof** – simple, linear.
- This is called proof-carrying authorization (PCA) [AF'99,Bau'03]

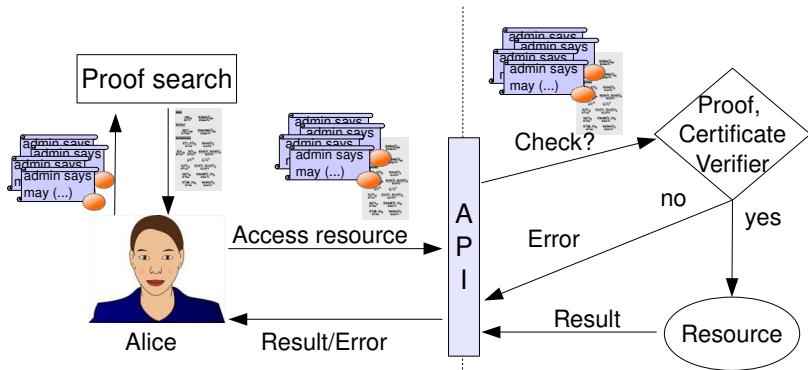
Background: From Representation to Enforcement

- Summary of policy representation in logic
 - ▶ Principals sign statements; represent as k says s
 - ▶ Statements together constitute the policy
$$\Gamma = \{k_1 \text{ says } s_1, \dots, k_n \text{ says } s_n\}$$
- How is this representation used for **enforcement** in the guard?
- **Idea!** Use **logical inference** for enforcement
- Allow access only if Γ entails a fixed formula s , i.e., $\Gamma \vdash s$
 - ▶ s may have the form $(\text{admin says } \text{may}(k, f, \text{read}))$
- Completely rigorous, formal
- Question: Isn't logical inference undecidable?
- Answer: Yes, so require that each *access request be accompanied by a proof of $\Gamma \vdash s$* .
 - ▶ The guard must only **verify the proof** – simple, linear.
- This is called **proof-carrying authorization (PCA)** [AF'99,Bau'03]

Background: From Representation to Enforcement

- Summary of policy representation in logic
 - ▶ Principals sign statements; represent as k says s
 - ▶ Statements together constitute the policy
$$\Gamma = \{k_1 \text{ says } s_1, \dots, k_n \text{ says } s_n\}$$
- How is this representation used for **enforcement** in the guard?
- Idea! Use **logical inference** for enforcement
- Allow access only if Γ entails a fixed formula s , i.e., $\Gamma \vdash s$
 - ▶ s may have the form $(\text{admin says } \text{may}(k, f, \text{read}))$
- Completely rigorous, formal
- Question: Isn't logical inference undecidable?
- Answer: Yes, so require that each *access request be accompanied by a proof of $\Gamma \vdash s$* .
 - ▶ The guard must only **verify the proof** – simple, linear.
- This is called proof-carrying authorization (PCA) [AF'99,Bau'03]

Background: Proof-Carrying Authorization (PCA)



- PCA has been implemented and tested for access to:

- ▶ Web services
- ▶ Physical devices (Grey@CyLab)

[Bau'03]

[BGM+'05]

Background: Summary of Relevant Past Work

- Logic (of some kinds) is a useful medium for representing authorization policies
- Policies represented in logic can be enforced using proofs (PCA)

Outline

- 1 Background
- 2 Overview of Thesis: Motivation and Contributions**
- 3 BL_S : A Logic for Static Authorizations
- 4 BL: A Logic for Dynamic Authorizations
- 5 PCFS: The File System
- 6 Conclusion

Motivation 1: Proof-Theoretic Foundations

- Proof theory is of paramount importance in authorization
 - ▶ Policies are interpreted by proofs, so logic should be **justified by proof theory** (proof-theoretic semantics)
 - ▶ Enforcement (PCA) is based in inference, so proof theory is the **foundation** of logic-based authorization
 - ▶ Proof theory is the basis of **practical tools** like theorem provers, which are central in the practice of PCA
- Surprisingly, despite several years of work, little investigation of proof theory of authorization logics

Investigate the proof theory of a new authorization logic, BL

Motivation 1: Proof-Theoretic Foundations

- Proof theory is of paramount importance in authorization
 - ▶ Policies are interpreted by proofs, so logic should be **justified by proof theory** (proof-theoretic semantics)
 - ▶ Enforcement (PCA) is based in inference, so proof theory is the **foundation** of logic-based authorization
 - ▶ Proof theory is the basis of **practical tools** like theorem provers, which are central in the practice of PCA
- Surprisingly, despite several years of work, little investigation of proof theory of authorization logics

Investigate the proof theory of a new authorization logic, BL

Motivation 2: Support for Dynamic Authorizations

- A **dynamic authorization** is an access permission that is not permanent
- A proof establishing access may be invalidated because:
 - ▶ Some hypothesis (certificate) **expires**
 - ▶ Some part of the **state changes**
 - ▶ Some **credential is consumed** elsewhere
 - ▶ Some credential is **revoked**
- Usually handled by requiring dynamic parts of proofs to be **constructed in the access protocol**, e.g., via nonces.

Include explicit time, state, and linearity in BL

Motivation 2: Support for Dynamic Authorizations

- A **dynamic authorization** is an access permission that is not permanent
- A proof establishing access may be invalidated because:
 - ▶ Some hypothesis (certificate) **expires**
 - ▶ Some part of the **state changes**
 - ▶ Some **credential is consumed** elsewhere
 - ▶ Some credential is **revoked**
- Usually handled by requiring dynamic parts of proofs to be **constructed in the access protocol**, e.g., via nonces.

Include explicit time, state, and linearity in BL

Motivation 2: Support for Dynamic Authorizations

- A **dynamic authorization** is an access permission that is not permanent
- A proof establishing access may be invalidated because:
 - ▶ Some hypothesis (certificate) **expires**
 - ▶ Some part of the **state changes**
 - ▶ Some **credential is consumed** elsewhere
 - ▶ Some credential is **revoked**
- Usually handled by requiring dynamic parts of proofs to be **constructed in the access protocol**, e.g., via nonces.

Include explicit time, state, and linearity in BL

Motivation 2: Support for Dynamic Authorizations

- A **dynamic authorization** is an access permission that is not permanent
- A proof establishing access may be invalidated because:
 - ▶ Some hypothesis (certificate) **expires**
 - ▶ Some part of the **state changes**
 - ▶ Some **credential is consumed** elsewhere
 - ▶ Some credential is **revoked**
- Usually handled by requiring dynamic parts of proofs to be **constructed in the access protocol**, e.g., via nonces.

Include explicit time, state, and linearity in BL

Motivation 3: Efficient Enforcement via Proofs

- In proof-carrying authorization the guard checks a proof at each access. How fast is this?
- Approx. 7ms (Grey), up to 100ms in thesis' case study
- Is it **fast enough for a file system? No!**
- What do we do? Separate policy decision (proof verification) from the guard
 - ▶ A trusted verifier checks proofs; issues cryptographic capabilities (procaps)
 - ▶ Procaps are checked by the guard
 - ▶ Non-trivial with dynamic authorizations
 - ★ Time, state, and consumable credentials must be checked by the guard, not the verifier. **Procaps are conditional.**

A new file system – PCFS – that uses proofs, and its correctness

Motivation 3: Efficient Enforcement via Proofs

- In proof-carrying authorization the guard checks a proof at each access. How fast is this?
- Approx. 7ms (Grey), up to 100ms in thesis' case study
- Is it **fast enough for a file system? No!**
- What do we do? Separate policy decision (proof verification) from the guard
 - ▶ A trusted verifier checks proofs; issues cryptographic capabilities (**procaps**)
 - ▶ Procaps are checked by the guard
 - ▶ Non-trivial with dynamic authorizations
 - ★ Time, state, and consumable credentials must be checked by the guard, not the verifier. **Procaps are conditional.**

A new file system – PCFS – that uses proofs, and its correctness

Summary of Work in the Thesis

- A new authorization logic BL, its proof theory, metatheory.
 - ▶ BL is a first-order modal logic
 - ▶ BL is intuitionistic
 - ▶ BL contains explicit time and system state
 - ▶ Proof verification procedure that extracts dynamic elements from a BL proof; theorem that extraction is sound and complete for verification
 - ▶ An extension BL^L supports linearity
 - ▶ Sound and complete goal-directed search for BL
- Design and implementation of the file system PCFS
 - ▶ Backwards compatible design using default procaps
 - ▶ Implementation of PCFS; measurements showing its feasibility in practice
- Investigation of expressiveness
 - ▶ Embeddings of GP logic and Soutei in BL
 - ▶ Case study of policies for controlling access to classified information in the U.S.

Contributions of the Thesis

- Proof theory for authorization logics, illustrated in a new logic BL; its applications – proof search, proof normalization
 - Logical treatment of dynamism in authorization policies, and a provably correct, efficient mechanism for their enforcement
 - Implementation of the enforcement mechanism in a file system, PCFS, demonstration of efficiency, and a case study to establish expressiveness.
- + Several minor, technical contributions

Outline

- 1 Background
- 2 Overview of Thesis: Motivation and Contributions
- 3 BL_S : A Logic for Static Authorizations**
- 4 BL: A Logic for Dynamic Authorizations
- 5 PCFS: The File System
- 6 Conclusion

BL_S: Syntax

- A fragment of BL; lacks explicit time and state
- Introduces proof theory and investigates the nature of “*k* says *s*”
- Syntax:

$$r, s ::= p \mid r \wedge s \mid r \vee s \mid r \supset s \mid \top \mid \perp \mid \forall x.s \mid \exists x.s \mid k \text{ says } s$$

- Intuitionistic, first-order quantifiers (standard interpretation)
- What is the meaning of *k* says *s*?
 - ▶ *k* says *s* has been interpreted differently in earlier authorization logics, e.g., \Box of modal logic K, and lax modality
 - ▶ BL_S has a new interpretation

BL_S: Syntax

- A fragment of BL; lacks explicit time and state
- Introduces proof theory and investigates the nature of “*k* says *s*”
- Syntax:

$$r, s ::= p \mid r \wedge s \mid r \vee s \mid r \supset s \mid \top \mid \perp \mid \forall x.s \mid \exists x.s \mid k \text{ says } s$$

- Intuitionistic, first-order quantifiers (standard interpretation)
- What is the meaning of *k* says *s*?
 - ▶ *k* says *s* has been interpreted differently in earlier authorization logics, e.g., \square of modal logic K, and lax modality
 - ▶ BL_S has a new interpretation

BL_S: Syntax

- A fragment of BL; lacks explicit time and state
- Introduces proof theory and investigates the nature of “ k says s ”
- Syntax:

$$r, s ::= p \mid r \wedge s \mid r \vee s \mid r \supset s \mid \top \mid \perp \mid \forall x.s \mid \exists x.s \mid k \text{ says } s$$

- Intuitionistic, first-order quantifiers (standard interpretation)
- What is the meaning of k says s ?
 - ▶ k says s has been interpreted differently in earlier authorization logics, e.g., \square of modal logic K, and lax modality
 - ▶ BL_S has a new interpretation

The Why and How of (k says s) in BL_S

- Motivation: express a common policy motif, called *exclusive delegation* in the thesis
- Requirement: principal k delegates to principal k' authority over a formula s , but retains no authority over s .
- payroll says $\forall x.((\text{HR says } \text{is_employee}(x)) \supset \text{may_be_paid}(x))$
(Payroll may not determine who an employee is)
- Requirements:
 1. Above formula and (HR says $\text{is_employee}(k)$) should imply (payroll says $\text{may_be_paid}(k)$)
 2. Above formula and (payroll says $\text{is_employee}(k)$) should **not** imply (payroll says $\text{may_be_paid}(k)$)
- Abstracting,
 3. (1) is equivalent to admitting (k says s) \supset k' says k says s
 4. (2) is violated if the following holds: (k says s) \supset k says k' says s
 - ★ In particular, a lax interpretation of (k says s) violates (2)

The Why and How of (k says s) in BL_S

- Motivation: express a common policy motif, called *exclusive delegation* in the thesis
- Requirement: principal k delegates to principal k' authority over a formula s , but retains no authority over s .
- payroll says $\forall x.((\text{HR says } \text{is_employee}(x)) \supset \text{may_be_paid}(x))$
(Payroll may not determine who an employee is)
- Requirements:
 1. Above formula and (HR says $\text{is_employee}(k)$) should imply (payroll says $\text{may_be_paid}(k)$)
 2. Above formula and (payroll says $\text{is_employee}(k)$) should **not** imply (payroll says $\text{may_be_paid}(k)$)
- Abstracting,
 3. (1) is equivalent to admitting (k says s) \supset k' says k says s
 4. (2) is violated if the following holds: (k says s) \supset k says k' says s
 - ★ In particular, a lax interpretation of (k says s) violates (2)

The Why and How of (k says s) in BL_S

- Motivation: express a common policy motif, called *exclusive delegation* in the thesis
- Requirement: principal k delegates to principal k' authority over a formula s , but retains no authority over s .
- payroll says $\forall x.((\text{HR says } \text{is_employee}(x)) \supset \text{may_be_paid}(x))$
(Payroll may not determine who an employee is)
- Requirements:
 1. Above formula and ($\text{HR says } \text{is_employee}(k)$) should imply ($\text{payroll says } \text{may_be_paid}(k)$)
 2. Above formula and ($\text{payroll says } \text{is_employee}(k)$) should **not** imply ($\text{payroll says } \text{may_be_paid}(k)$)
- Abstracting,
 3. (1) is equivalent to admitting $(k \text{ says } s) \supset k' \text{ says } k \text{ says } s$
 4. (2) is violated if the following holds: $(k \text{ says } s) \supset k \text{ says } k' \text{ says } s$
 - ★ In particular, a lax interpretation of $(k \text{ says } s)$ violates (2)

The Why and How of (k says s) in BL_S

- Motivation: express a common policy motif, called *exclusive delegation* in the thesis
- Requirement: principal k delegates to principal k' authority over a formula s , but retains no authority over s .
- payroll says $\forall x.((\text{HR says } \text{is_employee}(x)) \supset \text{may_be_paid}(x))$
(Payroll may not determine who an employee is)
- Requirements:
 1. Above formula and (HR says $\text{is_employee}(k)$) should imply (payroll says $\text{may_be_paid}(k)$)
 2. Above formula and (payroll says $\text{is_employee}(k)$) should **not** imply (payroll says $\text{may_be_paid}(k)$)
- Abstracting,
 3. (1) is equivalent to admitting (k says s) \supset k' says k says s
 4. (2) is violated if the following holds: (k says s) \supset k says k' says s
 - ★ In particular, a lax interpretation of (k says s) violates (2)

The Why and How of $(k \text{ says } s)$ in BL_S (Contd.)

- We have an upper and lower bound on the strength of $(k \text{ says } s)$
 - ▶ $\vdash ((k \text{ says } s) \supset k' \text{ says } k \text{ says } s)$
 - ▶ $\not\vdash ((k \text{ says } s) \supset k \text{ says } k' \text{ says } s)$
- Other constraints on $(k \text{ says } s)$
 - ▶ $\vdash (k \text{ says } (s \supset s')) \supset ((k \text{ says } s) \supset (k \text{ says } s'))$
 - ▶ $\not\vdash s \supset (k \text{ says } s)$ (This entails $((k \text{ says } s) \supset k \text{ says } k' \text{ says } s)$)
 - ▶ $\not\vdash (k \text{ says } s) \supset s$
- Above all, we want a nice proof theory!
- So, we think hard for many days, and get a suitable interpretation of $(k \text{ says } s)$, which we use in BL_S

The Why and How of $(k \text{ says } s)$ in BL_S (Contd.)

- We have an upper and lower bound on the strength of $(k \text{ says } s)$
 - ▶ $\vdash ((k \text{ says } s) \supset k' \text{ says } k \text{ says } s)$
 - ▶ $\not\vdash ((k \text{ says } s) \supset k \text{ says } k' \text{ says } s)$
- Other constraints on $(k \text{ says } s)$
 - ▶ $\vdash (k \text{ says } (s \supset s')) \supset ((k \text{ says } s) \supset (k \text{ says } s'))$
 - ▶ $\not\vdash s \supset (k \text{ says } s)$ (This entails $((k \text{ says } s) \supset k \text{ says } k' \text{ says } s)$)
 - ▶ $\not\vdash (k \text{ says } s) \supset s$
- Above all, we want a nice proof theory!

- So, we think hard for many days, and get a suitable interpretation of $(k \text{ says } s)$, which we use in BL_S

BL_S: Proof Theory (Sequent Calculus)

- Basic judgments:
 - ▶ s true: formula s is true
 - ▶ k claims s : k states or claims s (Internalized as k says s)
- Hypotheses $\Gamma ::= \cdot \mid \Gamma, s \text{ true} \mid \Gamma, k \text{ claims } s$
- Sequent $\Sigma; \Gamma \xrightarrow{k} s \text{ true}$
 - ▶ Σ is the sorting – ignored in this talk
 - ▶ k is called the view

Intuitively, “assuming that claims of k in Γ are true, s must hold”

BL_S: Proof Theory (Sequent Calculus)

- Rules for judgments and (k says s):

$$\frac{}{\Sigma; \Gamma, p \text{ true} \xrightarrow{k} p \text{ true}} \text{init} \qquad \frac{\Sigma; \Gamma, k \text{ claims } s, s \text{ true} \xrightarrow{k} s' \text{ true}}{\Sigma; \Gamma, k \text{ claims } s \xrightarrow{k} s' \text{ true}} \text{claims}$$

$$\frac{\Sigma; \Gamma \mid \xrightarrow{k} s \text{ true}}{\Sigma; \Gamma \xrightarrow{k'} (k \text{ says } s) \text{ true}} \text{saysR}$$

$$\frac{\Sigma; \Gamma, k \text{ says } s \text{ true}, k \text{ claims } s \xrightarrow{k'} s' \text{ true}}{\Sigma; \Gamma, k \text{ says } s \text{ true} \xrightarrow{k'} s' \text{ true}} \text{saysL}$$

- Rules for other connectives follow standard template, ignoring views

BL_S: Metatheoretic Properties

- Admissibility of cut: $\Sigma; \Gamma \xrightarrow{k} s$ true and $\Sigma; \Gamma, s \text{ true} \xrightarrow{k} s'$ true imply $\Sigma; \Gamma \xrightarrow{k} s'$ true
 - ▶ Cannot be proved by itself (using syntactic induction)
 - ▶ Add a second clause for claims:
 $\Sigma; \Gamma \mid \xrightarrow{k} s$ true and $\Sigma; \Gamma, k \text{ claims } s \xrightarrow{k'} s'$ true imply $\Sigma; \Gamma \xrightarrow{k'} s'$ true
 - ▶ Unusual – there is no relation between k and k'
- Identity: $\Sigma; \Gamma, s \text{ true} \xrightarrow{k} s$ true
- Consequences:
 - ▶ Consistency, i.e., $\not\vdash \perp$
 - ▶ Subformula property
 - ▶ Equivalence of natural deduction and sequent calculus (also axiomatic system)
 - ▶ Proof normalization
 - ▶ Correctness of embeddings – GP logic, Soutei, and CS4
- Other properties: view subsumption (unique to BL), structural properties, etc.

BL_S: Metatheoretic Properties

- Admissibility of cut: $\Sigma; \Gamma \xrightarrow{k} s$ true and $\Sigma; \Gamma, s \text{ true} \xrightarrow{k} s'$ true imply $\Sigma; \Gamma \xrightarrow{k} s'$ true
 - ▶ Cannot be proved by itself (using syntactic induction)
 - ▶ Add a second clause for claims:
 $\Sigma; \Gamma \mid \xrightarrow{k} s$ true and $\Sigma; \Gamma, k \text{ claims } s \xrightarrow{k'} s'$ true imply $\Sigma; \Gamma \xrightarrow{k'} s'$ true
 - ▶ Unusual – there is no relation between k and k'
- Identity: $\Sigma; \Gamma, s \text{ true} \xrightarrow{k} s$ true
- Consequences:
 - ▶ Consistency, i.e., $\not\vdash \perp$
 - ▶ Subformula property
 - ▶ Equivalence of natural deduction and sequent calculus (also axiomatic system)
 - ▶ Proof normalization
 - ▶ Correctness of embeddings – GP logic, Soutei, and CS4
- Other properties: view subsumption (unique to BL), structural properties, etc.

BL_S: Metatheoretic Properties

- Admissibility of cut: $\Sigma; \Gamma \xrightarrow{k} s$ true and $\Sigma; \Gamma, s \text{ true} \xrightarrow{k} s'$ true imply $\Sigma; \Gamma \xrightarrow{k} s'$ true
 - ▶ Cannot be proved by itself (using syntactic induction)
 - ▶ Add a second clause for claims:
 $\Sigma; \Gamma \mid \xrightarrow{k} s$ true and $\Sigma; \Gamma, k \text{ claims } s \xrightarrow{k'} s'$ true imply $\Sigma; \Gamma \xrightarrow{k'} s'$ true
 - ▶ Unusual – there is no relation between k and k'
- Identity: $\Sigma; \Gamma, s \text{ true} \xrightarrow{k} s$ true
- Consequences:
 - ▶ Consistency, i.e., $\not\vdash \perp$
 - ▶ Subformula property
 - ▶ Equivalence of natural deduction and sequent calculus (also axiomatic system)
 - ▶ Proof normalization
 - ▶ Correctness of embeddings – GP logic, Soutei, and CS4
- Other properties: view subsumption (unique to BL), structural properties, etc.

BL_S: Metatheoretic Properties

- Admissibility of cut: $\Sigma; \Gamma \xrightarrow{k} s$ true and $\Sigma; \Gamma, s \text{ true} \xrightarrow{k} s'$ true imply $\Sigma; \Gamma \xrightarrow{k} s'$ true
 - ▶ Cannot be proved by itself (using syntactic induction)
 - ▶ Add a second clause for claims:
 $\Sigma; \Gamma \mid \xrightarrow{k} s$ true and $\Sigma; \Gamma, k \text{ claims } s \xrightarrow{k'} s'$ true imply $\Sigma; \Gamma \xrightarrow{k'} s'$ true
 - ▶ Unusual – there is no relation between k and k'
- Identity: $\Sigma; \Gamma, s \text{ true} \xrightarrow{k} s$ true
- Consequences:
 - ▶ Consistency, i.e., $\not\vdash \perp$
 - ▶ Subformula property
 - ▶ Equivalence of natural deduction and sequent calculus (also axiomatic system)
 - ▶ Proof normalization
 - ▶ Correctness of embeddings – GP logic, Soutei, and CS4
- Other properties: view subsumption (unique to BL), structural properties, etc.

BL_S: Translations To and From

- Correctness of all embeddings based on simulation between sequent calculi proofs; rely on subformula property
- Embedding GP logic in BL_S: Assume a principal ℓ such that $(\ell \text{ says } s) \supset (k \text{ says } s)$

$$\begin{aligned}\lceil p \rceil &= \ell \text{ says } p \\ \lceil s_1 \supset s_2 \rceil &= \ell \text{ says } (\lceil s_1 \rceil \supset \lceil s_2 \rceil) \\ \lceil k \text{ says } s \rceil &= \ell \text{ says } k \text{ says } \lceil s \rceil\end{aligned}$$

- Embedding Soutei in BL_S: Soutei is a fragment, modulo syntactic changes
- Embedding propositional BL_S in CS4: Map all propositional logic connectives to themselves; $k \text{ says } s$ to $\Box(g(k) \supset \lceil s \rceil)$

Corollary: BL_S (hence also BL) is quite expressive

BL_S: Translations To and From

- Correctness of all embeddings based on simulation between sequent calculi proofs; rely on subformula property
- Embedding GP logic in BL_S: Assume a principal ℓ such that $(\ell \text{ says } s) \supset (k \text{ says } s)$

$$\begin{aligned}\lceil p \rceil &= \ell \text{ says } p \\ \lceil s_1 \supset s_2 \rceil &= \ell \text{ says } (\lceil s_1 \rceil \supset \lceil s_2 \rceil) \\ \lceil k \text{ says } s \rceil &= \ell \text{ says } k \text{ says } \lceil s \rceil\end{aligned}$$

- Embedding Soutei in BL_S: Soutei is a fragment, modulo syntactic changes
- Embedding propositional BL_S in CS4: Map all propositional logic connectives to themselves; $k \text{ says } s$ to $\Box(g(k) \supset \lceil s \rceil)$

Corollary: BL_S (hence also BL) is quite expressive

BL_S: Translations To and From

- Correctness of all embeddings based on simulation between sequent calculi proofs; rely on subformula property
- Embedding GP logic in BL_S: Assume a principal ℓ such that $(\ell \text{ says } s) \supset (k \text{ says } s)$

$$\begin{aligned}\lceil p \rceil &= \ell \text{ says } p \\ \lceil s_1 \supset s_2 \rceil &= \ell \text{ says } (\lceil s_1 \rceil \supset \lceil s_2 \rceil) \\ \lceil k \text{ says } s \rceil &= \ell \text{ says } k \text{ says } \lceil s \rceil\end{aligned}$$

- Embedding Soutei in BL_S: Soutei is a fragment, modulo syntactic changes
- Embedding propositional BL_S in CS4: Map all propositional logic connectives to themselves; $k \text{ says } s$ to $\Box(g(k) \supset \lceil s \rceil)$

Corollary: BL_S (hence also BL) is quite expressive

BL_S: Translations To and From

- Correctness of all embeddings based on simulation between sequent calculi proofs; rely on subformula property
- Embedding GP logic in BL_S: Assume a principal ℓ such that $(\ell \text{ says } s) \supset (k \text{ says } s)$

$$\begin{aligned}\lceil p \rceil &= \ell \text{ says } p \\ \lceil s_1 \supset s_2 \rceil &= \ell \text{ says } (\lceil s_1 \rceil \supset \lceil s_2 \rceil) \\ \lceil k \text{ says } s \rceil &= \ell \text{ says } k \text{ says } \lceil s \rceil\end{aligned}$$

- Embedding Soutei in BL_S: Soutei is a fragment, modulo syntactic changes
- Embedding propositional BL_S in CS4: Map all propositional logic connectives to themselves; $k \text{ says } s$ to $\Box(g(k) \supset \lceil s \rceil)$

Corollary: BL_S (hence also BL) is quite expressive

BL_S: Translations To and From

- Correctness of all embeddings based on simulation between sequent calculi proofs; rely on subformula property
- Embedding GP logic in BL_S: Assume a principal ℓ such that $(\ell \text{ says } s) \supset (k \text{ says } s)$

$$\begin{aligned}\lceil p \rceil &= \ell \text{ says } p \\ \lceil s_1 \supset s_2 \rceil &= \ell \text{ says } (\lceil s_1 \rceil \supset \lceil s_2 \rceil) \\ \lceil k \text{ says } s \rceil &= \ell \text{ says } k \text{ says } \lceil s \rceil\end{aligned}$$

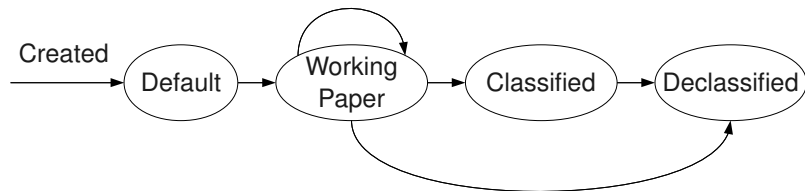
- Embedding Soutei in BL_S: Soutei is a fragment, modulo syntactic changes
- Embedding propositional BL_S in CS4: Map all propositional logic connectives to themselves; $k \text{ says } s$ to $\Box(g(k) \supset \lceil s \rceil)$

Corollary: BL_S (hence also BL) is quite expressive

Outline

- 1 Background
- 2 Overview of Thesis: Motivation and Contributions
- 3 BL_S : A Logic for Static Authorizations
- 4 BL: A Logic for Dynamic Authorizations**
- 5 PCFS: The File System
- 6 Conclusion

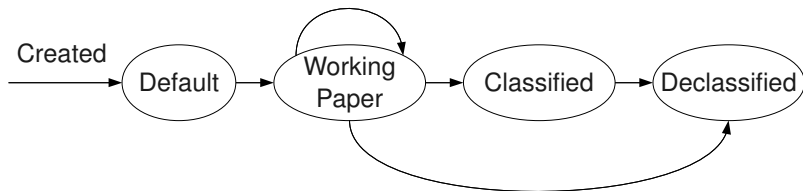
Dynamic Authorization: Motivating Example for State



Life cycle of a sensitive file in the intelligence community

- Access to a file depends on its status at the time of access
- In the actual system, status may be contained in the file meta-data or a database
- How do we represent the status in the logic?
- In BL, represent status directly via a special predicate **status**(f, s) that would be verified by checking the file meta-data or the database

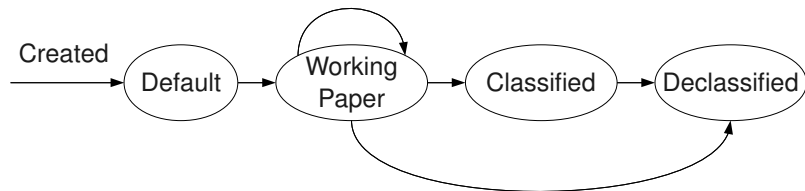
Dynamic Authorization: Motivating Example for State



Life cycle of a sensitive file in the intelligence community

- Access to a file depends on its status at the time of access
- In the actual system, status may be contained in the file meta-data or a database
- How do we represent the status in the logic?
- In BL, represent status directly via a special predicate **status**(f, s) that would be verified by checking the file meta-data or the database

Dynamic Authorization: Motivating Example for State



Life cycle of a sensitive file in the intelligence community

- Access to a file depends on its status at the time of access
- In the actual system, status may be contained in the file meta-data or a database
- How do we represent the status in the logic?
- In BL, represent status directly via a special predicate **status**(f, s) that would be verified by checking the file meta-data or the database

BL: Syntax

- Formulas:

$$s, t ::= \dots \mid k \text{ says } s \mid s @ [u_1, u_2] \mid c \mid i$$

- $s @ [u_1, u_2]$ means that s is true in interval $[u_1, u_2]$, but possibly not outside.
 - ▶ Used to represent time-bounded credentials, as in prior work [DGP'08]
- c denotes a constraint; $c ::= \dots \mid u_1 \leq u_2 \mid is(u, u')$
 - ▶ Necessary to draw useful consequences from time-dependent hypotheses, e.g., $s @ [2009:01:01, 2009:12:31]$ should imply $s @ [2009:10:01, 2009:10:01]$.
 - ▶ Constraints implemented as a decision procedure
- i is an interpreted predicate
 - ▶ Not established via certificates; represents a part of the system state via a trusted program
 - ▶ Useful for representing state-dependent authorizations

BL: Syntax

- Formulas:

$$s, t ::= \dots \mid k \text{ says } s \mid s @ [u_1, u_2] \mid c \mid i$$

- $s @ [u_1, u_2]$ means that s is true in interval $[u_1, u_2]$, but possibly not outside.

- ▶ Used to represent time-bounded credentials, as in prior work

[DGP'08]

- c denotes a constraint; $c ::= \dots \mid u_1 \leq u_2 \mid i_S(u, u')$

- ▶ Necessary to draw useful consequences from time-dependent hypotheses, e.g., $s @ [2009:01:01, 2009:12:31]$ should imply $s @ [2009:10:01, 2009:10:01]$.

- ▶ Constraints implemented as a decision procedure

- i is an interpreted predicate

- ▶ Not established via certificates; represents a part of the system state via a trusted program
- ▶ Useful for representing state-dependent authorizations

BL: Syntax

- Formulas:

$$s, t ::= \dots \mid k \text{ says } s \mid s @ [u_1, u_2] \mid c \mid i$$

- $s @ [u_1, u_2]$ means that s is true in interval $[u_1, u_2]$, but possibly not outside.
 - ▶ Used to represent time-bounded credentials, as in prior work [DGP'08]
- c denotes a constraint; $c ::= \dots \mid u_1 \leq u_2 \mid i_s(u, u')$
 - ▶ Necessary to draw useful consequences from time-dependent hypotheses, e.g., $s @ [2009:01:01, 2009:12:31]$ should imply $s @ [2009:10:01, 2009:10:01]$.
 - ▶ Constraints implemented as a decision procedure
- i is an interpreted predicate
 - ▶ Not established via certificates; represents a part of the system state via a trusted program
 - ▶ Useful for representing state-dependent authorizations

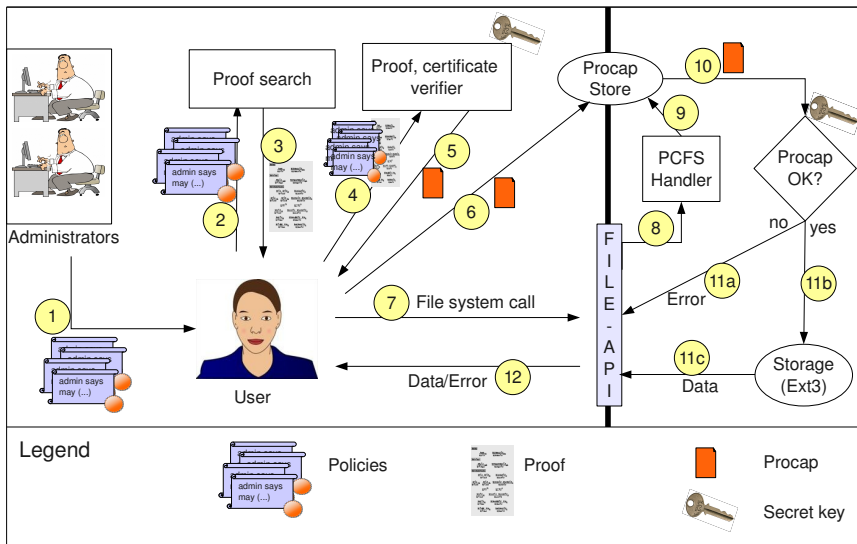
Summary of Dynamic Authorizations

- Represent authorization policies with dynamic elements in BL
 - ▶ Explicit time – $s @ [u_1, u_2]$, constraints
 - ▶ Dynamic state – interpreted predicates
 - ▶ Consumable credentials – linearity (extension BL^L)
- Proof theory and metatheoretic properties more complicated, but work out as expected; details in thesis

Outline

- 1 Background
- 2 Overview of Thesis: Motivation and Contributions
- 3 BL_S : A Logic for Static Authorizations
- 4 BL: A Logic for Dynamic Authorizations
- 5 PCFS: The File System**
- 6 Conclusion

The PCFS Architecture



Two-part Checking for Dynamic Authorizations

- Ideally, the proof verifier would like to verify that $\Sigma; \cdot; E; \Gamma \xrightarrow{\nu} M : s @ [u, u]$ is provable, where
 - ▶ u is the time when *the authorization is used for access*
 - ▶ E is the system state at time u
- Since proof verification precedes access, neither u , nor E can be known to the verifier, so what do we do?
- Idea! Perform a verification parametric in u and E
- Verifier checks that $\Sigma; \cdot; \cdot; \Gamma \xrightarrow{\nu} M : s @ [ctime, ctime]$
 - ▶ E is now empty
 - ▶ u is replaced by a fresh constant $ctime$
- During verification:
 - ▶ Constraint containing $ctime$ is encountered \Rightarrow write to $procap$ in a list \mathcal{C}
 - ▶ Interpreted predicate is encountered \Rightarrow write to $procap$ in a list \mathcal{I}
- \mathcal{C} and \mathcal{I} are **conditions** on the $procap$
- The guard checks $\models \mathcal{C}[u/ctime]$ and $E \models \mathcal{I}$ during access (it knows u and E)

Two-part Checking for Dynamic Authorizations

- Ideally, the proof verifier would like to verify that $\Sigma; \cdot; E; \Gamma \xrightarrow{\nu} M : s @ [u, u]$ is provable, where
 - ▶ u is the time when *the authorization is used for access*
 - ▶ E is the system state at time u
- Since proof verification precedes access, neither u , nor E can be known to the verifier, so what do we do?
- Idea! Perform a verification parametric in u and E
- Verifier checks that $\Sigma; \cdot; \cdot; \Gamma \xrightarrow{\nu} M : s @ [ctime, ctime]$
 - ▶ E is now empty
 - ▶ u is replaced by a fresh constant $ctime$
- During verification:
 - ▶ Constraint containing $ctime$ is encountered \Rightarrow write to $procap$ in a list \mathcal{C}
 - ▶ Interpreted predicate is encountered \Rightarrow write to $procap$ in a list \mathcal{I}
- \mathcal{C} and \mathcal{I} are **conditions** on the $procap$
- The guard checks $\models \mathcal{C}[u/ctime]$ and $E \models \mathcal{I}$ during access (it knows u and E)

Two-part Checking for Dynamic Authorizations

- Ideally, the proof verifier would like to verify that $\Sigma; \cdot; E; \Gamma \xrightarrow{\nu} M : s @ [u, u]$ is provable, where
 - ▶ u is the time when *the authorization is used for access*
 - ▶ E is the system state at time u
- Since proof verification precedes access, neither u , nor E can be known to the verifier, so what do we do?
- **Idea!** Perform a verification parametric in u and E
- Verifier checks that $\Sigma; \cdot; \cdot; \Gamma \xrightarrow{\nu} M : s @ [ctime, ctime]$
 - ▶ E is now empty
 - ▶ u is replaced by a fresh constant $ctime$
- During verification:
 - ▶ Constraint containing $ctime$ is encountered \Rightarrow write to $procap$ in a list \mathcal{C}
 - ▶ Interpreted predicate is encountered \Rightarrow write to $procap$ in a list \mathcal{I}
- \mathcal{C} and \mathcal{I} are **conditions** on the $procap$
- The guard checks $\models \mathcal{C}[u/ctime]$ and $E \models \mathcal{I}$ during access (it knows u and E)

Two-part Checking for Dynamic Authorizations

- Ideally, the proof verifier would like to verify that $\Sigma; \cdot; E; \Gamma \xrightarrow{\nu} M : s @ [u, u]$ is provable, where
 - ▶ u is the time when *the authorization is used for access*
 - ▶ E is the system state at time u
- Since proof verification precedes access, neither u , nor E can be known to the verifier, so what do we do?
- Idea! Perform a verification parametric in u and E
- Verifier checks that $\Sigma; \cdot; \cdot; \Gamma \xrightarrow{\nu} M : s @ [ctime, ctime]$
 - ▶ E is now empty
 - ▶ u is replaced by a fresh constant $ctime$
- During verification:
 - ▶ Constraint containing $ctime$ is encountered \Rightarrow write to $procap$ in a list \mathcal{C}
 - ▶ Interpreted predicate is encountered \Rightarrow write to $procap$ in a list \mathcal{I}
- \mathcal{C} and \mathcal{I} are **conditions** on the $procap$
- The guard checks $\models \mathcal{C}[u/ctime]$ and $E \models \mathcal{I}$ during access (it knows u and E)

Two-part Checking for Dynamic Authorizations

- Ideally, the proof verifier would like to verify that $\Sigma; \cdot; E; \Gamma \xrightarrow{\nu} M : s @ [u, u]$ is provable, where
 - ▶ u is the time when *the authorization is used for access*
 - ▶ E is the system state at time u
- Since proof verification precedes access, neither u , nor E can be known to the verifier, so what do we do?
- Idea! Perform a verification parametric in u and E
- Verifier checks that $\Sigma; \cdot; \cdot; \Gamma \xrightarrow{\nu} M : s @ [ctime, ctime]$
 - ▶ E is now empty
 - ▶ u is replaced by a fresh constant $ctime$
- During verification:
 - ▶ Constraint containing $ctime$ is encountered \Rightarrow write to $procap$ in a list \mathcal{C}
 - ▶ Interpreted predicate is encountered \Rightarrow write to $procap$ in a list \mathcal{I}
- \mathcal{C} and \mathcal{I} are **conditions** on the $procap$
- The guard checks $\models \mathcal{C}[u/ctime]$ and $E \models \mathcal{I}$ during access (it knows u and E)

Two-part Checking for Dynamic Authorizations (Contd.)

- Summarily, proof checking is a two-part procedure:
 - ▶ Logical structure in the proof verifier, ahead of access
 - ▶ Dynamic dependencies in the guard
- Thesis formalizes extraction of constraint and interpreted predicates; proves soundness and completeness
- Soundness: If the two part checking of a proof succeeds, then the proof is valid at the time of access
- Completeness: If the user presents a proof valid at the time of access to the verifier, then:
 - ▶ The verifier's parametric verification will succeed
 - ▶ The conditions produced will check at the time of access
- Conclusion: The revised architecture is as rigorous as PCA, but scales better

Two-part Checking for Dynamic Authorizations (Contd.)

- Summarily, proof checking is a two-part procedure:
 - ▶ Logical structure in the proof verifier, ahead of access
 - ▶ Dynamic dependencies in the guard
- Thesis formalizes extraction of constraint and interpreted predicates; proves soundness and completeness
- Soundness: If the two part checking of a proof succeeds, then the proof is valid at the time of access
- Completeness: If the user presents a proof valid at the time of access to the verifier, then:
 - ▶ The verifier's parametric verification will succeed
 - ▶ The conditions produced will check at the time of access
- Conclusion: The revised architecture is as rigorous as PCA, but scales better

Two-part Checking for Dynamic Authorizations (Contd.)

- Summarily, proof checking is a two-part procedure:
 - ▶ Logical structure in the proof verifier, ahead of access
 - ▶ Dynamic dependencies in the guard
- Thesis formalizes extraction of constraint and interpreted predicates; proves soundness and completeness
- Soundness: If the two part checking of a proof succeeds, then the proof is valid at the time of access
- Completeness: If the user presents a proof valid at the time of access to the verifier, then:
 - ▶ The verifier's parametric verification will succeed
 - ▶ The conditions produced will check at the time of access
- Conclusion: The revised architecture is as rigorous as PCA, but scales better

Two-part Checking for Dynamic Authorizations (Contd.)

- Summarily, proof checking is a two-part procedure:
 - ▶ Logical structure in the proof verifier, ahead of access
 - ▶ Dynamic dependencies in the guard
- Thesis formalizes extraction of constraint and interpreted predicates; proves soundness and completeness
- Soundness: If the two part checking of a proof succeeds, then the proof is valid at the time of access
- Completeness: If the user presents a proof valid at the time of access to the verifier, then:
 - ▶ The verifier's parametric verification will succeed
 - ▶ The conditions produced will check at the time of access
- Conclusion: The revised architecture is as rigorous as PCA, but scales better

Two-part Checking for Dynamic Authorizations (Contd.)

- Summarily, proof checking is a two-part procedure:
 - ▶ Logical structure in the proof verifier, ahead of access
 - ▶ Dynamic dependencies in the guard
- Thesis formalizes extraction of constraint and interpreted predicates; proves soundness and completeness
- Soundness: If the two part checking of a proof succeeds, then the proof is valid at the time of access
- Completeness: If the user presents a proof valid at the time of access to the verifier, then:
 - ▶ The verifier's parametric verification will succeed
 - ▶ The conditions produced will check at the time of access
- Conclusion: The revised architecture is as rigorous as PCA, but scales better

Implementation of PCFS

- **Factored into two parts**
- Front end: Certificates, proof search, proof verifier
 - ▶ Command line tools, written in SML
 - ▶ Efficiency is not a concern
- Back end: Procap store, procap checking, file access
 - ▶ Built using FUSE, implemented as a C++ process server that handles kernel upcalls
 - ▶ Efficiency is a primary concern; measurements in the thesis
- Other aspects of PCFS, not related to logic
 - ▶ Backwards compatibility
 - ▶ Finer permissions, can represent DAC and MAC
 - ▶ Protection of secret key and configuration files

Implementation of PCFS

- Factored into two parts
- Front end: Certificates, proof search, proof verifier
 - ▶ Command line tools, written in SML
 - ▶ Efficiency is not a concern
- Back end: Procap store, procap checking, file access
 - ▶ Built using FUSE, implemented as a C++ process server that handles kernel upcalls
 - ▶ Efficiency is a primary concern; measurements in the thesis
- Other aspects of PCFS, not related to logic
 - ▶ Backwards compatibility
 - ▶ Finer permissions, can represent DAC and MAC
 - ▶ Protection of secret key and configuration files

Implementation of PCFS

- Factored into two parts
- Front end: Certificates, proof search, proof verifier
 - ▶ Command line tools, written in SML
 - ▶ Efficiency is not a concern
- Back end: Procap store, procap checking, file access
 - ▶ Built using FUSE, implemented as a C++ process server that handles kernel upcalls
 - ▶ Efficiency is a primary concern; measurements in the thesis
- Other aspects of PCFS, not related to logic
 - ▶ Backwards compatibility
 - ▶ Finer permissions, can represent DAC and MAC
 - ▶ Protection of secret key and configuration files

Implementation of PCFS

- Factored into two parts
- Front end: Certificates, proof search, proof verifier
 - ▶ Command line tools, written in SML
 - ▶ Efficiency is not a concern
- Back end: Procap store, procap checking, file access
 - ▶ Built using FUSE, implemented as a C++ process server that handles kernel upcalls
 - ▶ Efficiency is a primary concern; measurements in the thesis
- Other aspects of PCFS, not related to logic
 - ▶ Backwards compatibility
 - ▶ Finer permissions, can represent DAC and MAC
 - ▶ Protection of secret key and configuration files

Outline

- 1 Background
- 2 Overview of Thesis: Motivation and Contributions
- 3 BL_S : A Logic for Static Authorizations
- 4 BL: A Logic for Dynamic Authorizations
- 5 PCFS: The File System
- 6 Conclusion

Conclusion

“Logic, grounded in strong **proof theory**, can be used for representing and reasoning about **dynamic** authorization policies, and for **efficiently** enforcing them.”

Thank You!

BL: Proof Theory

- Represent constraint domain(s) and state predicates as judgments without rules:
 - ▶ $\Sigma; \Psi \models c$ (Ψ = assumed constraints)
 - ▶ $\Sigma; E \models i$ (E = abstract representation of system state)
- Basic judgments:
 - ▶ $s \circ [u_1, u_2]$: s true during $[u_1, u_2]$
 - ▶ k claims $s \circ [u_1, u_2]$: k claims s true during $[u_1, u_2]$
- Sequents: $\Sigma; \Psi; E; \Gamma \xrightarrow{k, u_b, u_e} s \circ [u_1, u_2]$
- Rules based on prior work [DGP'08]
- Right rules for interpreted predicates and constraints

$$\frac{\Sigma; \Psi \models c}{\Sigma; \Psi; E; \Gamma \xrightarrow{\nu} c \circ [u_1, u_2]} \text{consR}$$

$$\frac{\Sigma; E \models i}{\Sigma; \Psi; E; \Gamma \xrightarrow{\nu} i \circ [u_1, u_2]} \text{interR}$$

- Metatheoretic properties are much more complicated, but hold as expected