Logics for Access Control and Security

Morteza Amini
m_amini@ce.sharif.edu

Network Security Center
Department of Computer Engineering
Sharif University of Technology

16th October 2006
**Introduction**
- The Problem
- Why Logics?
- Applications of Logics in Security

**A Calculus for Access Control in DS**
- Basic Concepts
- The Basic Logic
- Roles and Delegation
- Extensions
- Access Control Decision Algorithm

**A Propositional Policy Algebra for Access Control**
- The Problem
- Basic Concepts
- Syntax
- Semantics
- The Algebra of Operators
- Determinism, Consistency, and Completeness

**Summary and Conclusions**

**References**
1 Introduction
   - The Problem
   - Why Logics?
   - Applications of Logics in Security

2 A Calculus for Access Control in DS
   - Basic Concepts
   - The Basic Logic
   - Roles and Delegation
   - Extensions
   - Access Control Decision Algorithm

3 A Propositional Policy Algebra for Access Control
   - The Problem
   - Basic Concepts
   - Syntax
   - Semantics
   - The Algebra of Operators
   - Determinism, Consistency, and Completeness

4 Summary and Conclusions

5 References
1. Introduction
   - The Problem
   - Why Logics?
   - Applications of Logics in Security

2. A Calculus for Access Control in DS
   - Basic Concepts
   - The Basic Logic
   - Roles and Delegation
   - Extensions
   - Access Control Decision Algorithm

3. A Propositional Policy Algebra for Access Control
   - The Problem
   - Basic Concepts
   - Syntax
   - Semantics
   - The Algebra of Operators
   - Determinism, Consistency, and Completeness

4. Summary and Conclusions

5. References
1 Introduction
- The Problem
- Why Logics?
- Applications of Logics in Security

2 A Calculus for Access Control in DS
- Basic Concepts
- The Basic Logic
- Roles and Delegation
- Extensions
- Access Control Decision Algorithm

3 A Propositional Policy Algebra for Access Control
- The Problem
- Basic Concepts
- Syntax
- Semantics
- The Algebra of Operators
- Determinism, Consistency, and Completeness

4 Summary and Conclusions

5 References
1 Introduction
   - The Problem
   - Why Logics?
   - Applications of Logics in Security

2 A Calculus for Access Control in DS
   - Basic Concepts
   - The Basic Logic
   - Roles and Delegation
   - Extensions
   - Access Control Decision Algorithm

3 A Propositional Policy Algebra for Access Control
   - The Problem
   - Basic Concepts
   - Syntax
   - Semantics
   - The Algebra of Operators
   - Determinism, Consistency, and Completeness

4 Summary and Conclusions

5 References
1 Introduction
- The Problem
- Why Logics?
- Applications of Logics in Security

2 A Calculus for Access Control in DS
- Basic Concepts
- The Basic Logic
- Roles and Delegation
- Extensions
- Access Control Decision Algorithm

3 A Propositional Policy Algebra for Access Control
- The Problem
- Basic Concepts
- Syntax
- Semantics
- The Algebra of Operators
- Determinism, Consistency, and Completeness

4 Summary and Conclusions

5 References
The Problem

Three ingredients are essential for security in computing systems:

1. A trusted computing system
2. Authentication
3. Authorization

The ingredients are fairly understood in centralized systems. What about in Distributed Systems?

Difficulties with DS:

- Scale
- Communication
- Booting
- Loading
- Authentication
- Authorization
Why Logics?

1. Clean Foundation: hence formal guarantees
2. Flexibility
3. Expressiveness: possible to describe protocols & policies at reasonable level of abstraction.
4. Independency from implementation: important in heterogeneous distributed systems
5. Declarativeness: Users not required any programming ability
6. Having Ability of Verification
7. ...
Applications of Logics in Security

1. Logic-based Policy Specification
2. Policy Composition Frameworks
   - Constraint-based Approach
   - Deontic Approach
   - Algebraic Approach
3. Policy Evaluation and Verification
4. Trust Management
5. Model Checking for Protocol Verification
6. Intrusion Detection
### Basic Concepts

#### Principal

Participants in a distributed systems are called *agents*, and the symbols that represent agents in logical expressions are called *principals*.

- Users and Machines
- Channels
- Conjunction of principals \((A \land B)\)
- Groups
- Principals in roles \((A \text{ as } R)\)
- Principals on behalf of principals \((B \text{ for } A) \lor (B \mid A)\)
Basic Concepts

**$A \land B$:** $A$ and $B$ as cosigners. A request from $A \land B$ is a request that both $A$ and $B$ make.

**$A \lor B$:** represents a group of which $A$ and $B$ are the sole members.

**$A as R$:** a principal $A$ in role $R$.

**$B|A$:** the principal obtained when $B$ speaks on behalf of $A$, not necessarily with a proof that $A$ has delegated authority to $B$. We pronounce it $B$ quoting $A$.

$B|A$ says $s$ if $B$ says that $A$ says $s$. 
Basic Concepts

$A \wedge B$: $A$ and $B$ as cosigners. A request from $A \wedge B$ is a request that both $A$ and $B$ make.

$A \lor B$: represents a group of which $A$ and $B$ are the sole members.

$A$ as $R$: a principal $A$ in role $R$.

$B \mid A$: the principal obtained when $B$ speaks on behalf of $A$, not necessarily with a proof that $A$ has delegated authority to $B$. We pronounce it $B$ quoting $A$.

$B \mid A$ says $s$ if $B$ says that $A$ says $s$. 
**Basic Concepts**

\( A \land B \): A and B as cosigners. A request from \( A \land B \) is a request that both \( A \) and \( B \) make.

\( A \lor B \): represents a group of which \( A \) and \( B \) are the sole members.

**A as R:** a principal \( A \) in role \( R \).

\( B\mid A \): the principal obtained when \( B \) speaks on behalf of \( A \), not necessarily with a proof that \( A \) has delegated authority to \( B \). We pronounce it \( B \) quoting \( A \).

\( B\mid A \) says \( s \) if \( B \) says that \( A \) says \( s \).
Basic Concepts

\(A \land B\): \(A\) and \(B\) as cosigners. A request from \(A \land B\) is a request that both \(A\) and \(B\) make.

\(A \lor B\): represents a group of which \(A\) and \(B\) are the sole members.

\(A as R\): a principal \(A\) in role \(R\).

\(B|A\): the principal obtained when \(B\) speaks on behalf of \(A\), not necessarily with a proof that \(A\) has delegated authority to \(B\). We pronounce it \(B\) quoting \(A\).

\(B|A\) says \(s\) if \(B\) says that \(A\) says \(s\).
Basic Concepts

\( B \text{ for } A \): the principal obtained when \( B \) speaks on behalf of \( A \) with appropriate delegation certificates.

\( B \text{ for } A \) says \( s \) when \( A \) has delegated authority to \( B \) and \( B \) says that \( A \) says \( s \).

\( A \Rightarrow B \) (\( A \) implies \( B \)) or (\( A \) speaks for \( B \)): \( A \) is a member of group \( B \). \( A \) is at least as powerful as \( B \).

\[ A \Rightarrow B \text{ iff } A = A \land B. \]

\( A \text{ says } s \): says is a modal operator. \( A \text{ says } s \) is a formula that means the principal \( A \) believes that formula \( s \) is true.

\( \supset \): logical implication.
Basic Concepts

**B for A:** the principal obtained when B speaks on behalf of A with appropriate delegation certificates.  
*B for A says s* when A has delegated authority to B and B says that A says s.

\[ A \Rightarrow B \] (A implies B) or (A speaks for B): A is a member of group B. A is at least as powerful as B.

\[ A \Rightarrow B \iff A = A \land B. \]

**A says s:** says is a modal operator. A says s is a formula that means the principal A believes that formula s is true.

\[ \vdash \]: logical implication.
Basic Concepts

*B for A*: the principal obtained when *B* speaks on behalf of *A* with appropriate delegation certificates.
*B for A says s* when *A* has delegated authority to *B* and *B* says that *A* says *s*.

*A ⇒ B* (*A* implies *B*) or (*A* speaks for *B*): *A* is a member of group *B*. *A* is at least as powerful as *B*.

\[ A \Rightarrow B \text{ iff } A = A \land B. \]

*A says s*: says is a modal operator. *A says s* is a formula that means the principal *A* believes that formula *s* is true.

⊃: logical implication.
Basic Concepts

*B for A*: the principal obtained when *B* speaks on behalf of *A* with appropriate delegation certificates.
*B for A says s* when *A* has delegated authority to *B* and *B* says that *A* says *s*.

*A ⇒ B* (*A implies B*) or (*A speaks for B*): *A* is a member of group *B*. *A* is at least as powerful as *B*.

\[ A \Rightarrow B \text{ iff } A = A \land B. \]

*A says s*: says is a modal operator. *A says s* is a formula that means the principal *A* believes that formula *s* is true.

⊃: logical implication.
**A controls s:** \((A \text{ says } s) \supset s\).

**ACL** is a list of assertions like \((A \text{ controls } s)\). If \(s\) is clear, ACL is a list of principals trusted on \(s\).

**Corollary**

\[
\begin{align*}
B \text{ controls } s \land A \Rightarrow B \\
\therefore A \text{ controls } s
\end{align*}
\]
**Basic Concepts**

*A controls s*: \((A \text{ says } s) \supset s\).

**ACL** is a list of assertions like \((A \text{ controls } s)\). If \(s\) is clear, ACL is a list of principals trusted on \(s\).

**Corollary**

\[
\frac{B \text{ controls } s \land A \Rightarrow B}{\therefore A \text{ controls } s}
\]
The Basic Logic
A Calculus of Principals

Some Axioms

1. $\land$ and $|$ are primitive operators of calculus of principals.
2. $\land$ is associative, commutative, and idempotent
   - Principals form a semilattice under $\land$.
3. $|$ is associative.
   - Principals form a semigroup under $|$.
4. $|$ distributes over $\land$.
   - Multiplicativity implies monotonicity.

Corollary

Principals structure multiplicative semilattice semigroup, which is isomorphism with binary relations with union and composition.
The Basic Logic
A Calculus of Principals

Some Axioms

1. $\land$ and $|$ are primitive operators of calculus of principals.
2. $\land$ is associative, commutative, and idempotent
   - Principals form a semilattice under $\land$.
3. $|$ is associative.
   - Principals form a semigroup under $|$.  
4. $|$ distributes over $\land$.
   - Multiplicativity implies monotonicity.

Corollary

Principals structure multiplicative semilattice semigroup, which is isomorphism with binary relations with union and composition.
Syntax

The **formulas** are defined inductively as follows:

1. a countable supply of primitive propositions $p_0, p_1, \ldots$ are formulas;
2. if $s$ and $s'$ are formulas then so are $\neg s$ and $s \land s'$;
3. if $A$ and $B$ are principal expressions then $A \Rightarrow B$ is a formula;
4. if $A$ is a principal expression and $s$ is a formula then $A$ says $s$ is a formula.
The Basic Logic

A Logic of Principals and Their Statements

Axioms

1. The basic axioms for normal modal logic:
   - if $s$ is an instance of a propositional-logic tautology then $\vdash s$;
   - if $\vdash s$ and $\vdash (s \supset s')$ then $\vdash s'$;
   - $\vdash A \text{ says } (s \supset s') \supset (A \text{ says } s \supset A \text{ says } s')$;
   - if $\vdash s$ then $\vdash A \text{ says } s$, for every $A$.

2. The axioms of the calculus of principals:
   - if $s$ is a valid formula of the calculus of principals then $\vdash s$.

3. The axioms connect the calculus to the modal logic:
   - $\vdash (A \land B) \text{ says } s \equiv (A \text{ says } s) \land (B \text{ says } s)$;
   - $\vdash (B|A) \text{ says } s \equiv B \text{ says } A \text{ says } s$;
   - $\vdash (A \Rightarrow B) \supset ((A \text{ says } s) \supset (B \text{ says } s))$.
   - which is equivalent to $(A = B) \supset ((A \text{ says } s) \equiv (B \text{ says } s))$. 
Kripke Semantics

A structure $\mathcal{M}$ is a tuple $\langle \mathcal{W}, \omega_0, \mathcal{I}, \mathcal{J} \rangle$, where:

- $\mathcal{W}$ is a set (a set of possible worlds)
- $\omega_0$ is distinguished element of $\mathcal{W}$
- $\mathcal{I}$ is an interpretation function
  \[ \mathcal{I} : PropositionSymbols \rightarrow \mathcal{P}(\mathcal{W}). \]
  ($\mathcal{I}(s)$ is a set of worlds where the proposition symbol is true)
- $\mathcal{J}$ is an interpretation function
  \[ \mathcal{J} : Principals \rightarrow \mathcal{P}(\mathcal{W} \times \mathcal{W}) \]
The Basic Logic

Semantics

Example

- $l$ agent is in produce department
- $\neg l$ agent is in meat department
- $b$ the bananas are yellow
- $\neg b$ the bananas are green
- $p$ the pork is fresh
- $\neg p$ the pork is spoiled
The meaning function $\mathcal{R}$ extends $\mathcal{I}$ as follows:

- $\mathcal{R}(A_i) = \mathcal{I}(A_i)$
- $\mathcal{R}(A \land B) = \mathcal{R}(A) \cup \mathcal{R}(B)$
- $\mathcal{R}(A|B) = \mathcal{R}(A) \circ \mathcal{R}(B)$
The Basic Logic

Semantics

The meaning function $\mathcal{E}$ extends $\mathcal{I}$ as follows:

- $\mathcal{E}(p_i) = \mathcal{I}(p_i)$
- $\mathcal{E}(\neg s) = \mathcal{W} - \mathcal{E}(s)$
- $\mathcal{E}(s \land s') = \mathcal{E}(s) \cap \mathcal{E}(s')$
- $\mathcal{E}(A \text{ says } s) = \{\omega | R(A)(\omega) \subseteq \mathcal{E}(s)\}$
- $\mathcal{E}(A \Rightarrow B) = \mathcal{W}$ if $R(B) \subseteq R(A)$ and $\emptyset$ otherwise
  
  $R(C)(\omega) = \{\omega' | \omega R(C)\omega'\}$
Soundness

The axioms are sound, in the sense that if \( \vdash s \) then \( \models s \).

Completeness

Although useful for our application, the axioms are not complete. For example, the formula
\[
C \text{ says } (A \Rightarrow B)) \equiv ((A \Rightarrow B) \lor (C \text{ says false}))
\]
is valid but not provable.
The Basic Logic

Semantics

**Soundness**
The axioms are sound, in the sense that if $\vdash s$ then $\models s$.

**Completeness**
Although useful for our application, the axioms are not complete. For example, the formula

$$C \text{ says } (A \Rightarrow B)) \equiv ((A \Rightarrow B) \lor (C \text{ says false}))$$

is valid but not provable.
The Basic Logic
On Idempotence

- The idempotence of $|$ and $\forall$ is intuitively needed.
  - $A|A = A$: $A$ says $A$ says $s$ and $A$ says $s$ are equal.
  - $G|A$ in an ACL postulates $G|G|A$ and $G|G \Rightarrow G$.
  - Idempotence impose more complexity. e.g., it yields $(A \land B) \Rightarrow (B|A)$. On a request of $A \land B$ we need to check both $(A|B)$ and $(B|A)$.

- We unable to find a sensible condition on binary relations that would force idempotence and would be preserved by union and composition.

**Corollary**

The authors preferred to do without idempotence and rely on assumptions of the form $G|G \Rightarrow G$. 
The idempotence of | and for is intuitively needed.

- \( A|A = A \): A says A says s and A says s are equal.
- \( G|A \) in an ACL postulates \( G|G|A \) and \( G|G \Rightarrow G \).
- Idempotence impose more complexity. e.g., it yields \( (A \land B) \Rightarrow (B|A) \). On a request of \( A \land B \) we need to check both \( (A|B) \) and \( (B|A) \).
- We unable to find a sensible condition on binary relations that would force idempotence and would be preserved by union and composition.

Corollary

The authors preferred to do without idempotence and rely on assumptions of the form \( G|G \Rightarrow G \).
Roles

What Roles Are For?

- "Least Privilege" principle.
- For diminishing power of a user (A as R).
- For limiting untrusted software.

Roles, Groups, and Resources

- Roles may be related to Groups. e.g., G_{role} related to group G. A as G_{role} means A act in the role of member of G.
- We do allow roles related to groups but this relation is not formal.
- Roles may correspond to a set of resources.
Roles

What Roles Are For?

- "Least Privilege" principle.
- For diminishing power of a user (A as R).
- For limiting untrusted software.

Roles, Groups, and Resources

- Roles may be related to Groups. e.g., $G_{role}$ related to group $G$. A as $G_{role}$ means A act in the role of member of $G$.
- We do allow roles related to groups but this relation is not formal.
- Roles may correspond to a set of resources.
Roles
The Encoding

**Definition (Role)**

In the binary relation model, roles are subsets of the identity relations: $1 \Rightarrow R$.

A principal $A$ in role $R$ is defined as $A$ as $R$ which is equal to $A|R$.

A special principal $1$, the *identity*, believes everything that is true and nothing that is not.

$$R(1)(\omega) = \omega, \; \forall \omega \in \mathcal{W}$$
Roles

The Encoding

Roles reduce privileges.

$$\mathcal{R}(A) \circ \mathcal{R}(R_1) \subseteq \mathcal{R}(A)$$

An arbitrary principal relation $\mathcal{R}(A)$ . . .  
\[ ∘ \]
... composed with a role relation $\mathcal{R}(R)$ . . .  
\[ = \]
... gives a new relation that is always a subset of $\mathcal{R}(A)$.  

M. Amini  
Logics for Access Control and Security
<table>
<thead>
<tr>
<th>Role Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Monotonicity, multiplicativity, and associativity</strong> of</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Roles
The Encoding

Role Properties

- All roles are:
  - idempotent \((R | R = R)\)
  - commute with one another \((R | R' = R' | R)\).

These yield the following:

- \( A \text{ as } R \text{ as } R = A \text{ as } R \)
- \( A \text{ as } R \text{ as } R' = A \text{ as } R' \text{ as } R \)

- We assume:
  \( A \Rightarrow (A \text{ as } R) \) for all \( A \).
Delegation

Definition (Delegation)

The ability of a principal $A$ to give to another principal $B$ the authority to act on $A$’s behalf.

- Different mechanisms embody the concept of delegation in different settings.
- Delegation relies on authentication. It is often convenient to view delegation as independent.
- We consider 3 instances of delegations:
  - Delegation without certificates;
  - Delegation with certificates;
  - Delegation without delegate authentication.
**Delegation**

**Definition (Delegation)**

The ability of a principal $A$ to give to another principal $B$ the authority to act on $A$’s behalf.

- Different mechanisms embody the concept of delegation in different settings.
- Delegation relies on authentication. It is often convenient to view delegation as independent.
- We consider 3 instances of delegations:
  - Delegation without certificates;
  - Delegation with certificates;
  - Delegation without delegate authentication.
Delegation
The Forms of Delegation

Assumptions for Delegation

- Synchronized clocks are available.
- All principals can perform digital signature.
- The formula $K \text{ says } X$ represents the certificate $X$ encrypted under $K^{-1}$.
- A certificate authority $S$ provides certificates for the principals’ public keys.
1. Delegation Without Certificate

\[ B \text{ sends the signed request along with } A\text{'s name: } K_B \text{ says } A \text{ says } r \]

\[ \Rightarrow \]

When \( C \) receives \( r \), must believe that \( K_S \) is \( S\)'s public key: \( K_S \Rightarrow S \)

\[ \Rightarrow \]

\( C \) obtains a certificate encrypted under the inverse of \( K_S \): \( K_S \) says \((K_B \Rightarrow B)\)

\[ \Rightarrow \]

These yield: \( S \) says \((K_B \Rightarrow B)\)
Delegation
The Forms of Delegation

1. Delegation Without Certificate

B sends the signed request along with A’s name: $K_B$ says A says $r$

$\Downarrow$

When C receives $r$, must believe that $K_S$ is S’s public key: $K_S \Rightarrow S$

$\Downarrow$

C obtains a certificate encrypted under the inverse of $K_S$: $K_S$ says $(K_B \Rightarrow B)$

$\Downarrow$

These yield: S says $(K_B \Rightarrow B)$
1. Delegation Without Certificate

B sends the signed request along with A’s name: \( K_B \) says A says \( r \)

\[ \Downarrow \]

When \( C \) receives \( r \), must believe that \( K_S \) is S’s public key: \( K_S \Rightarrow S \)

\[ \Downarrow \]

\( C \) obtains a certificate encrypted under the inverse of \( K_S \): \( K_S \) says \( (K_B \Rightarrow B) \)

\[ \Downarrow \]

These yield: \( S \) says \( (K_B \Rightarrow B) \)
1. Delegation Without Certificate

*B* sends the signed request along with *A*’s name: \( K_B \text{ says } A \text{ says } r \)

\[ \Downarrow \]

When *C* receives *r*, must believe that \( K_S \) is *S*’s public key: \( K_S \Rightarrow S \)

\[ \Downarrow \]

*C* obtains a certificate encrypted under the inverse of \( K_S \): \( K_S \text{ says } (K_B \Rightarrow B) \)

\[ \Downarrow \]

These yield: \( S \text{ says } (K_B \Rightarrow B) \)
C trusts S for such statement: \textit{S controls} (K_B \Rightarrow B)

We obtain that C has B’s key: K_B \Rightarrow B

C sees a message as: K_B says A says r.

C obtains: B says A says r = (B \mid A) says r.

C checks ACL for r. If B \mid A exist in the ACL, r access is granted.
C trusts S for such statement: \( S \text{ controls } (K_B \Rightarrow B) \)

We obtain that C has B’s key: \( K_B \Rightarrow B \)

C sees a message as: \( K_B \text{ says } A \text{ says } r \).

C obtains: \( B \text{ says } A \text{ says } r = (B|A) \text{ says } r \).

C checks ACL for \( r \). If \( B|A \) exist in the ACL, \( r \) access is granted.
C trusts S for such statement: \( S \) controls \( (K_B \Rightarrow B) \)

We obtain that \( C \) has \( B \)'s key: \( K_B \Rightarrow B \)

\( C \) sees a message as: \( K_B \) says \( A \) says \( r \).

\( C \) obtains: \( B \) says \( A \) says \( r = (B|A) \) says \( r \).

\( C \) checks ACL for \( r \). If \( B|A \) exist in the ACL, \( r \) access is granted.
C trusts S for such statement: \( S \) controls \((K_B \Rightarrow B)\)

We obtain that C has B’s key: \( K_B \Rightarrow B \)

C sees a message as: \( K_B \) says A says r.

C obtains: B says A says r = \((B|A)\) says r.

C checks ACL for r. If B|A exist in the ACL, r access is granted.
C trusts S for such statement: \( S \ controls \ (K_B \Rightarrow B) \)

\[\downarrow\]

We obtain that C has B’s key: \( K_B \Rightarrow B \)

\[\downarrow\]

C sees a message as: \( K_B \ says \ A \ says \ r \).

\[\downarrow\]

C obtains: \( B \ says \ A \ says \ r = (B|A) \ says \ r \).

\[\downarrow\]

C checks ACL for \( r \). If \( B|A \) exist in the ACL, \( r \) access is granted.
Delegation
The Forms of Delegation

It is preferable for $A$ to issue a delegation certificate that proves the delegation to $B$.

2. Delegation With Certificate

After mutual authentication, $A$ issues a certificate to $B$ under $A$’s key: $K_A$ says ($B$ serves $A$)

$\downarrow$

Checking of public-key certificates from $S$ yields:
$A$ says ($B$ serves $A$)
Delegation
The Forms of Delegation

It is preferable for $A$ to issue a delegation certificate that proves the delegation to $B$.

2. Delegation With Certificate

After mutual authentication, $A$ issues a certificate to $B$ under $A$’s key: $K_A$ says ($B$ serves $A$)

\[\Downarrow\]

Checking of public-key certificates from $S$ yields:
$A$ says ($B$ serves $A$)
If C trusts A on this statement, C gets: 
\(((B|A) \text{ says } r) \land (B \text{ serves } A)\)

In our theories, this implies: \((B \text{ for } A) \text{ says } r\)

C checks ACL for \(r\). If \(B \text{ for } A\) exist in the ACL, \(r\) access is granted.
If C trusts A on this statement, C gets:

\(( (B|A) \text{ says } r ) \land (B \text{ serves } A)\)

⇓

In our theories, this implies: \((B \text{ for } A) \text{ says } r\)

⇓

C checks ACL for \(r\). If \(B \text{ for } A\) exist in the ACL, \(r\) access is granted.
If $C$ trusts $A$ on this statement, $C$ gets:

$$((B|A)\text{ says } r) \land (B\text{ serves } A)$$

In our theories, this implies: $(B\text{ for } A)\text{ says } r$

$C$ checks ACL for $r$. If $B\text{ for } A$ exist in the ACL, $r$ access is granted.
Omitting the authentication between $B$ and $C$, leaving the responsibility of authenticating $B$ solely to $A$.

3. Delegation Without Delegate Authentication

After mutual authentication, $A$ issues a certificate to $B$ under $A$’s key. The cert. includes a key $K_d$ and $B$’s name. [$K_d^{-1}$ is secret key]

$\Rightarrow$

$B$ can present $A$’s certificate to $C$ along with request under the delegation key $K_d$.

$\Rightarrow$

$C$ knows that $B$ has requested $r$ on behalf of $A$. Now $C$ checks the ACL for $r$. 

M. Amini Logics for Access Control and Security
Omitting the authentication between $B$ and $C$, leaving the responsibility of authenticating $B$ solely to $A$.

### 3. Delegation Without Delegate Authentication

After mutual authentication, $A$ issues a certificate to $B$ under $A$’s key. The cert. includes a key $K_d$ and $B$’s name. [$K_d^{-1}$ is secret key]

\[ \downarrow \]

$B$ can present $A$’s certificate to $C$ along with request under the delegation key $K_d$.

\[ \downarrow \]

$C$ knows that $B$ has requested $r$ on behalf of $A$. Now $C$ checks the ACL for $r$. 

M. Amini

Logics for Access Control and Security
Delegation
The Forms of Delegation

Omitting the authentication between $B$ and $C$, leaving the responsibility of authenticating $B$ solely to $A$.

3. Delegation Without Delegate Authentication

After mutual authentication, $A$ issues a certificate to $B$ under $A$’s key. The cert. includes a key $K_d$ and $B$’s name. [$K_d^{-1}$ is secret key]

$\Downarrow$

$B$ can present $A$’s certificate to $C$ along with request under the delegation key $K_d$.

$\Downarrow$

$C$ knows that $B$ has requested $r$ on behalf of $A$. Now $C$ checks the ACL for $r$. 
Delegation Properties

- for is monotonic
- for is multiplicative. This follows:
  \((B \land B') \text{ for } (A \land A') = (B \text{ for } A) \land (B \text{ for } A') \land (B' \text{ for } A) \land (B' \text{ for } A')\)
- \(B \text{ for } A\) is always defined, even if \(A\) has not delegated to \(B\). We have:
  - \((B | A) \land (C \text{ for } A) \Rightarrow ((B \land C) \text{ for } A)\)
  - \((B | A) \land (C \text{ for } A) \Rightarrow (B \text{ for } A)\)
Delegation Properties

- For possesses a weak associativity property:
  \[(C \text{ for } (B \text{ for } A)) \Rightarrow (C \text{ for } B) \text{ for } A\]

- \[(A \land (B\mid A)) \Rightarrow (B \text{ for } A), \text{ because } \mid \text{ is multiplicative.}\]

- If \(A = A\mid A\) then \(A = A \text{ for } A\), the idempotence of \(\mid\) implies the idempotence of for: \(A \text{ for } A = A\).
Extensions

The Basic logic can be extended in many ways, such as:

1. Adding Intersection
2. Adding Subtraction
3. Adding Variables
An intersection operation $\cap$ permits the construction of groups from groups.

- It can only applied to atomic symbols. $(A_j | A_i) \cap A_k$ is not valid.
- Conjunction is strictly weaker than intersection: $(A \cap B) \Rightarrow (A \land B)$, $(A \land B) \nRightarrow (A \cap B)$.
- **Application:** restricting access to only a particular member of a group.
  - The ACL entry $A \land G$ grants access to
    - $A$, if $A$ is a member of $G$,
    - $A \land B$, if $A$ and $B$ are members of $G$.
  - In contrast, $A \cap G$ just grant access to
    - $A$, if $A$ is a member of $G$. 
Group subtraction (−) provides the ability to specify that certain named individuals or subgroups within a supergroup should be denied access.

- $G'' - G$ means all members of $G''$, except for those members of $G''$ only via $G$.
- In distributed systems nonmembership may not be available.
- It is not very useful in distributed systems, just for groups which are managed by centralized subsystems.
Adding variables can increase the expressiveness, but more complexity.

- Variables can be included in ACLs.
- **Example:** Existing \((y|x)\) controls \(s\) in an ACL, results in granting access to \((B|A)\), by matching
  - \(x\) with \(A\),
  - \(y\) with \(B\).
Access Control Decision Algorithm

- The calculus of principals is undecidable.
- There are decidable variant of this calculus.
The Parts of an Instance of A.C. Decision Problem

- A principal $P$ that is making the request.
  [An expression in the calculus of principals]

- A statement $s$ represents what is being requested or asserted.
  [The service provider does not need to derive it logically from other statements.]

- Assumptions state implications among principals.
  [Assumptions about group membership (e.g., $P_i \Rightarrow G_i$) and idempotence (e.g., $G|G \Rightarrow G$)]

- Roles $R_0, R_1, ...$
  [Certain atomic symbols which may be obvious from their name]

- An ACL for $s$.
  [A list of expressions $E_0, E_1, ...$ of principals that are trusted on $s$.]
Access Control Decision Algorithm

A General Access Control Problem

The Basic Problem of Access Control

Deciding whether by having

- $\bigwedge_i (P_i \Rightarrow G_i)$, derived from the assumptions
- $\bigwedge_i (E_i \text{ controls } s)$, derived from the ACL

can we imply $P \text{ controls } s$. 
Outline

1. **Introduction**
   - The Problem
   - Why Logics?
   - Applications of Logics in Security

2. **A Calculus for Access Control in DS**
   - Basic Concepts
   - The Basic Logic
   - Roles and Delegation
   - Extensions
   - Access Control Decision Algorithm

3. **A Propositional Policy Algebra for Access Control**
   - The Problem
   - Basic Concepts
   - Syntax
   - Semantics
   - The Algebra of Operators
   - Determinism, Consistency, and Completeness

4. **Summary and Conclusions**

5. **References**
The Problem

Problem

- Information are governed by Policies.
- When information is shared, it is necessary compare, contrast, and compare the underlying security policies.
- Problems arise in doing so is diversity and incompatibility of
  - requirements,
  - security models,
  - security policies,
  - enforcement mechanisms.

Proposed Solution

Presenting a policy composition framework at the propositional level for access control.
The Problem

Information are governed by Policies.

When information is shared, it is necessary compare, contrast, and compare the underlying security policies.

Problems arise in doing so is diversity and incompatibility of

- requirements,
- security models,
- security policies,
- enforcement mechanisms.

Proposed Solution

Presenting a policy composition framework at the propositional level for access control.
Basic Concepts

**Definition (Permission and Permission Set)**

- **A permission** is an ordered pair \((object, \pm action)\).
  - Example: \((file1, +read)\), or \((file1, -write)\)
- **A permission set** is a set of such permissions.
  - Example: \(\{(file1, +read), (file1, -write)\}\)
Basic Concepts

Definition (Permission and Permission Set)

- A permission is an ordered pair \((object, \pm action)\).
  Example: \((file1, + read)\), or \((file1, - write)\)

- A permission set is a set of such permissions.
  Example: \\{\((file1, + read), (file1, - write)\)\}
Basic Concepts

Definition (Nondeterministic Transformers)

- Permission Set Transformation: \((s, \text{PermSet}) \mapsto (s, \text{PermSet}')\)
- Nondeterministic Transformers of Permission Sets: \((s, \text{SetOfPermSet}) \mapsto (s, \text{SetOfPermSet}')\)

Example:
\((A, \emptyset) \mapsto (A, \{(f1, +r), (f1, -w)\}, \{(f1, -r), (f1, +w)\})\)

It allows \(A\) two choices:
- \(\{(f1, +r), (f1, -w)\}\)
- \(\{(f1, -r), (f1, +w)\}\)
Basic Concepts

Definition (Policy)

A policy is interpreted as nondeterministic transformers on permission set assignments to subjects. Operations on policies are interpreted as relational or set theoretical operations on such nondeterministic transformers.
Internal vs. External Operator

There are two types of policy composition operators. Suppose

\[ P_1 = (C_1, \emptyset) \mapsto (C_1, \{(\text{check}, +\text{read}), (\text{check}, +\text{write})\}) \] and

\[ P_2 = (C_1, \emptyset) \mapsto (C_1, \{(\text{check}, +\text{read}), (\text{check}, +\text{approval})\}) \].

1. **Internal Operators:** E.g., internal union,

\[ P_1 \cup P_2 = (C_1, \emptyset) \mapsto (C_1, \{(\text{check}, +\text{read}), (\text{check}, +\text{write}), (\text{check}, +\text{approval})\}). \]

2. **External Operators:** E.g., external union,

\[ P_1 \sqcup P_2 = (C_1, \emptyset) \mapsto (C_1, \{\{(\text{check}, +\text{read}), (\text{check}, +\text{write})\},\{(\text{check}, +\text{read}), (\text{check}, +\text{approval})\}\}). \]
There are two types of policy composition operators. Suppose
\[ P_1 = (C_1, \emptyset) \mapsto (C_1, \{(\text{check}, +\text{read}), (\text{check}, +\text{write})\}) \] and
\[ P_2 = (C_1, \emptyset) \mapsto (C_1, \{(\text{check}, +\text{read}), (\text{check}, +\text{approval})\}) \].

1. **Internal Operators**: E.g., internal union,
\[ P_1 \cup P_2 = (C_1, \emptyset) \mapsto (C_1, \{((\text{check}, +\text{read})), (\text{check}, +\text{write}), (\text{check}, +\text{approval}))\}). \]

2. **External Operators**: E.g., external union,
\[ P_1 \sqcup P_2 = (C_1, \emptyset) \mapsto (C_1, \{\{(\text{check}, +\text{read})), (\text{check}, +\text{write})\}, \{(\text{check}, +\text{read})), (\text{check}, +\text{approval})\})}). \]
Basic Concepts

Individual vs. Set Proposition

For conditional authorization we have two types of propositions:

1. **Individual Proposition**: applies to an object.
   E.g., *Can read any check with a face value greater than $10,000.*

2. **Set Proposition**: applies to a set of objects.
   E.g., *if the total value of all checks to be read by a clerk is more than $10,000.*
Basic Concepts

Individual vs. Set Proposition

For **conditional authorization** we have two types of propositions:

1. **Individual Proposition**: applies to an object.
   
   E.g., *Can read any check with a face value greater than $10,000.*

2. **Set Proposition**: applies to a set of objects.
   
   E.g., *if the total value of all checks to be read by a clerk is more than $10,000.*
Syntax

Definition

- \( P_{atomic} \) is a terminal symbol taken from a set of atomic policies \( POL \)
- \( \phi_{atomic} \) is a terminal symbol taken from a set of atomic propositions \( PROP \)
- \( \Phi_{atomic} \) is a terminal symbol taken from a set of atomic set (second order) propositions \( SETP \)

Definition of policies, propositions and set propositions:

\[
\begin{align*}
P & := P_{atomic} | P \cup P | P \cap P | P \sqcup P | \phi \sqcup P | (\phi :: P) | (P \sqcap \phi) | P \cup P | P \cap P | P - P | \neg P | (\phi : P) | (p \sqcap \phi) | \circ P | P ; P | P^* | \\
\text{min}(P) | \text{max}(P) | oCom(P) | cCom(P) \\
\phi & := \phi_{atomic} | \phi \land \phi | \phi \lor \phi | \neg \phi \\
\Phi & := \Phi_{atomic} | \Phi \land \Phi | \Phi \lor \Phi | \neg \Phi
\end{align*}
\]
Syntax
Introduction to the Notations

- □ → external union
- □ → external intersection
- □ → external difference
- ↖ → external negation
- φ :: → external scoping
- ⊕φ → external provisioning
- ; → sequential composition
- * → closure (extension of sequential composition)
- ∪ → internal union
Syntax
Introduction to the Notations

- \( \cap \) \( \rightsquigarrow \) internal intersection
- \( - \) \( \rightsquigarrow \) internal difference
- \( \neg \) \( \rightsquigarrow \) internal negation
- \( \phi : \) \( \rightsquigarrow \) internal scoping
- \( \phi \) \( \rightsquigarrow \) internal provisioning
- \( \circ \) \( \rightsquigarrow \) invalidate permissions
- \( min, max \) \( \rightsquigarrow \) for conflict resolution, denial/permission take precedence
- \( oCom, cCom \) \( \rightsquigarrow \) open and close policy
Definition (Subjects, Objects, and Permissions)

1. **Subjects**: Let $S = \{s_i, i \in \mathbb{N}\}$ be a set of subjects.

2. **Objects**: Let $O = \{o_i, i \in \mathbb{N}\}$ be a set of objects.

3. **Signed Actions**: Let $A = \{a_i, i \in \mathbb{N}\}$ be a set of action terms. Then $A^\pm = A^+ \cup A^-$, where $A^+ = \{+a : a \in A\}$ and $A^- = \{-a : a \in A\}$ is the set of signed action terms.

4. **Roles**: Let $R = \{R_i, i \in \mathbb{N}\}$ be a set of roles.
Semantics
Basic Building Blocks of the Semantics

Definition (Subjects, Objects, and Permissions)

1) **Subjects**: Let $S = \{s_i, i \in \mathbb{N}\}$ be a set of subjects.

2) **Objects**: Let $O = \{o_i, i \in \mathbb{N}\}$ be a set of objects.

3) **Signed Actions**: Let $A = \{a_i, i \in \mathbb{N}\}$ be a set of action terms. Then $A^\pm = A^+ \cup A^-$, where $A^+ = \{+a : a \in A\}$ and $A^- = \{-a : a \in A\}$ is the set of signed action terms.

4) **Roles**: Let $\mathcal{R} = \{R_i, i \in \mathbb{N}\}$ be a set of roles.
Definition (Subjects, Objects, and Permissions)

1. **Subjects**: Let \( S = \{ s_i, i \in \mathbb{N} \} \) be a set of subjects.
2. **Objects**: Let \( O = \{ o_i, i \in \mathbb{N} \} \) be a set of objects.
3. **Signed Actions**: Let \( A = \{ a_i, i \in \mathbb{N} \} \) be a set of action terms. Then \( A^\pm = A^+ \cup A^- \), where \( A^+ = \{ +a : a \in A \} \) and \( A^- = \{ -a : a \in A \} \) is the set of signed action terms.
4. **Roles**: Let \( R = \{ R_i, i \in \mathbb{N} \} \) be a set of roles.
Definition (Subjects, Objects, and Permissions)

(1) **Subjects**: Let $S = \{s_i, i \in \mathbb{N}\}$ be a set of subjects.

(2) **Objects**: Let $O = \{o_i, i \in \mathbb{N}\}$ be a set of objects.

(3) **Signed Actions**: Let $\mathcal{A} = \{a_i, i \in \mathbb{N}\}$ be a set of action terms. Then $\mathcal{A}^\pm = \mathcal{A}^+ \cup \mathcal{A}^-$, where $\mathcal{A}^+ = \{+a : a \in \mathcal{A}\}$ and $\mathcal{A}^- = \{-a : a \in \mathcal{A}\}$ is the set of signed action terms.

(4) **Roles**: Let $\mathcal{R} = \{R_i, i \in \mathbb{N}\}$ be a set of roles.
Definition (Subjects, Objects, and Permissions)

(5) **Authorizations**: $(s, \text{PermSet})$ is an authorization if one of the following conditions holds:
- $s$ is either a subject or a role and $\text{PermSet} \subseteq \mathcal{O} \times \mathcal{A}^\pm$
- $s$ is a subject and $\text{PermSet}$ is a role. The notation $\mathcal{AU}(S, R, \mathcal{O}, \mathcal{A})$ denotes the set of all authorizations.

(6) **Permission-Prohibition Triples**: $(s, o, \pm a)$ where $s \in S, o \in \mathcal{O}, a \in \mathcal{A}^\pm$. The notation $\mathcal{T}(S, R, \mathcal{O}, \mathcal{A})$ denotes the set of all permission-prohibition triples.
**Definition (Subjects, Objects, and Permissions)**

(5) **Authorizations**: \((s, \text{PermSet})\) is an authorization if one of the following conditions holds:
- \(s\) is either a subject or a role and \(\text{PermSet} \subseteq O \times A^\pm\)
- \(s\) is a subject and \(\text{PermSet}\) is a role. The notation \(AU(S, R, O, A)\) denotes the set of all authorizations.

(6) **Permission-Prohibition Triples**: \((s, o, \pm a)\) where \(s \in S, o \in O, a \in A^\pm\). The notation \(T(S, R, O, A)\) denotes the set of all permission-prohibition triples.
**Definition (State)**

A state is a pair of mappings \((M_{prop}, M_{setProp})\), where

\[ M_{prop} : PROP \leftrightarrow \mathcal{P}(T) \] and \[ M_{setProp} : SETP \leftrightarrow \mathcal{P}(\mathcal{P}(T)) \].

**Definition (Interpreting Atomic Policies)**

An interpretation of atomic policies \(M_{AtPolicy}\) is a mapping from

\[ STATES \times POL \times (S \cup R) \times \mathcal{P}(O \times A^{\pm}) \leftrightarrow (S \cup R) \times \mathcal{P}(\mathcal{P}(O \times A^{\pm})) \]

satisfying the condition that \(s' = s\) for any \((s', PermSet') \in M_{AtPolicy}(St)(p)(s, PermSet)\).
Semantics

Basic Building Blocks of the Semantics

Definition (State)
A state is a pair of mappings \( (M_{prop}, M_{setProp}) \), where \( M_{prop} : PROP \mapsto \mathcal{P}(T) \) and \( M_{setProp} : SETP \mapsto \mathcal{P}(\mathcal{P}(T)) \).

Definition (Interpreting Atomic Policies)
An interpretation of atomic policies \( M_{AtPolicy} \) is a mapping from \( STATES \times \mathcal{P}OL \times (S \cup R) \times \mathcal{P}(O \times \mathcal{A}^\pm) \mapsto (S \cup R) \times \mathcal{P}(\mathcal{P}(O \times \mathcal{A}^\pm)) \) satisfying the condition that \( s' = s \) for any \( (s', PermSet') \in M_{AtPolicy}(St)(p)(s, PermSet) \).
Definition (Negating Permissions Sets)

If $PS \subseteq \mathcal{O} \times \mathcal{A}_\pm$ is a permission set, then

$-PS = \{(o, -a) : (o, +a) \in PS\} \cup \{(o, +a) : (o, -a) \in PS\}$.

If $r \in \mathcal{R}$ is a role, then $(o, -a) \in -r$ iff $(o, +a) \in r$ and $(o, +a) \in -r$ iff $(o, -a) \in r$. 
Semantics
Interpreting Policy Operators

Definition (Interpreting Policy Operators)

An interpretation $M_{AtPolicy}$ of atomic policies is extended to an interpretation $M_{policy}$ nonatomic policies using the following definition:

1. $M_{policy}(St)(p) = M_{AtPolicy}(St)(p)$ for all atomic policies $p$ and states $St$.
2. $M_{policy}(St)(p \sqcup q)(s, PermSet) = M_{policy}(St)(p)(s, PermSet) \cup M_{policy}(St)(q)(s, PermSet)$.
3. $M_{policy}(St)(p \sqcap q)(St)(s, PermSet) = M_{policy}(St)(p)(s, PermSet) \cap M_{policy}(St)(q)(s, PermSet)$.
4. $M_{policy}(St)(p \triangleright q)(s, PermSet) = M_{policy}(St)(p)(s, PermSet) \setminus M_{policy}(St)(q)(s, PermSet)$.
(5) \[ M_{\text{policy}}(St)(\uparrow p)(s, \text{PermSet}) = \{ (s, PS) : PS \in \mathcal{P}S \} \setminus M_{\text{policy}}(St)(p)(s, \text{PermSet}). \]

(6) \[ M_{\text{policy}}(St)(\phi :: p)(s, \text{PermSet}) = M_{\text{policy}}(St)(p)(s, \text{PermSet}), \text{if } (s, \text{PermSet}) \in M_{\text{setProp}}(St)(\phi), \text{and } \emptyset \text{ otherwise.} \]

(7) \[ M_{\text{policy}}(St)(p)(p || \phi)(s, \text{PermSet}) = M_{\text{prop}}(St)(p)(s, \text{PermSet}) \cap M_{\text{setProp}}(St)(\phi). \]

(8) \[ M_{\text{policy}}(St)(p; q)(s, \text{PermSet}) = \{ (s, \text{PermSet}_1) \in M_{\text{policy}}(St')(q)(s, \text{PermSet}_2) : \text{for some } (s, \text{PermSet}_2) \in M_{\text{policy}}(St)(p)(s, \text{PermSet}) \}. \]

(9) To define \( M_{\text{policy}}(St)(p^*)(s, \text{PermSet}) \), inductively define \( M_{\text{policy}}(St)(p^n) \) using the following rules:

(a) \[ M_{\text{policy}}(St)(p^1) = M_{\text{policy}}(St)(p). \]

(b) \[ M_{\text{policy}}(St)(p^{n+1}) = M_{\text{policy}}(St)((p; p^n) \cup p^n). \]

(c) \[ M_{\text{policy}}(St)(p^*) = \bigcup_{n \in \omega} M_{\text{policy}}(St)(p^n). \]

(10) \[ M_{\text{policy}}(St)(p \cup q)(s, \text{PermSet}) = \{ (s, \text{PermSet}_p \cup \text{PermSet}_q) : (s, \text{PermSet}_p) \in M_{\text{policy}}(St)(p)(s, \text{PermSet}_p) \text{ and } (s, \text{PermSet}_q) \in M_{\text{policy}}(St)(q)(s, \text{PermSet}_q) \}. \]
Semantics
Interpreting Policy Operators

(11) \( M_{policy}(St)(p \cap q)(s, PermSet) = \{ (s, PermSet_p \cap PermSet_q) : (s, PermSet_p) \in M_{policy}(St)(p)(s, PermSet_p) \) and \( (s, PermSet_q) \in M_{policy}(St)(q)(s, PermSet_q) \} \)

(12) \( M_{policy}(St)(p - q)(s, PermSet) = \{ (s, PermSet_p \setminus PermSet_q) : (s, PermSet_p) \in M_{policy}(St)(p)(s, PermSet_p) \) and \( (s, PermSet_q) \in M_{policy}(St)(q)(s, PermSet_q) \} \)

(13) \( M_{policy}(St)(\neg p)(s, PermSet) = \{ (s, \neg PermSet') : (s, PermSet) \in M_{policy}(St)(p)(s, PermSet') \} \).

(14) \( M_{policy}(St)(\phi : p)(s, PermSet\{(o, a) \notin M_{prop}(St)(\phi)\}) = M \) if \( M_{policy}(St)(p)(s, PermSet) = M \).

(15) \( M_{policy}(St)(p \setminus \phi)(s, PermSet) = M \) provided that \( M_{policy}(St)(p)(s, PermSet) = \{ (s, PermSet'\{(o, a) : (s, o, a) \notin M_{prop}(St')(\phi)\}) \} \), where \( M_{policy}(St)(p) = St' \).

(16) \( M_{policy}(St)(\max(p))(s, PermSet) = \{ (s, PermSet_1) : PermSet_1 = PermSet_2 \setminus \{ (o, -a) : (o, +a), (o, -a) \in PermSet_2 \} \) for some \( (s, PermSet_2) \in M_{policy}(St)(p)(s, PermSet) \).
Semantics
Interpreting Policy Operators

(17) \( M_{policy}(St)(\text{min}(p))(s, \text{PermSet}) = \{(s, \text{PermSet}_1) : \text{PermSet}_1 = \text{PermSet}_2 \setminus \{(o, +a) : (o, +a), (o, -a) \in \text{PermSet}_2\} \text{ for some } (s, \text{PermSet}_2) \in M_{policy}(St)(p)(s, \text{PermSet})\} \).

(18) \( M_{policy}(St)(\odot(p))(s, \text{PermSet}) = (s, \emptyset) \).

(19) \( M_{policy}(St)(\text{cCom}(p))(St)(s, \text{PermSet}) = \{(s, \text{PermSet}_1) : \text{PermSet}_1 = \text{PermSet}_2 \cup \{(o, -a) : (o, -a), (o, +a) \notin \text{PermSet}_2\} \text{ for some } (s, \text{PermSet}_2) \in M_{policy}(St)(p)(s, \text{PermSet})\} \).

(20) \( M_{policy}(St)(\text{oComp})(s, \text{PermSet}) = \{(s, \text{PermSet}_1) : \text{PermSet}_1 = \text{PermSet}_2 \cup \{(o, +a) : (o, -a), (o, +a) \notin \text{PermSet}_2\} \text{ for some } (s, \text{PermSet}_2) \in M_{policy}(St)(p)(s, \text{PermSet})\}. \)
The Algebra of Operators
The Algebra of External Operators

**Theorem (1)**

External operator namely, \((\mathcal{POL}, \sqcup, \sqcap, \neg, 1_{pol}, 0_{pol})\) form a Boolean algebra under the following interpretation of \(1_{pol}\) and \(0_{pol}\)

\[
M_{policy}(St)(1_{pol})(s, PermSet) = (s, \mathcal{P}(PS))
\]
\[
M_{policy}(St)(0_{pol})(s, PermSet) = (s, \emptyset)
\]

for each \(s \in S \cup R\) and \(St \in STATES\).

In summary, theorem 1 says that we can manipulate policy operators just as operators in propositional logic and therefore use the same disjunctive or conjunctive normal forms etc.
The Algebra of Operators

The Algebra of External Operators

Theorem (1)

External operator namely, \((\mathcal{P}OL, \sqcup, \sqcap, \neg, 1_{\text{pol}}, 0_{\text{pol}})\) form a Boolean algebra under the following interpretation of \(1_{\text{pol}}\) and \(0_{\text{pol}}\)

\[
M_{\text{policy}}(St)(1_{\text{pol}})(s, \text{PermSet}) = (s, \mathcal{P}(\text{PS}))
\]

\[
M_{\text{policy}}(St)(0_{\text{pol}})(s, \text{PermSet}) = (s, \emptyset)
\]

for each \(s \in S \cup R\) and \(St \in \text{STATES}\).

In summary, theorem 1 says that we can manipulate policy operators just as operators in propositional logic and therefore use the same disjunctive or conjunctive normal forms etc.
The Algebra of Operators
The Algebra of External Operators

Theorem (2)

Let $p, p_1, p_2,$ and $p_3$ be policy terms and
$I = (M_{policy}, M_{prop}, M_{setProp})$ be an interpretation. Then the following properties are valid in $I$:

1. **Idempotency of conjunctions and disjunctions:**
   - $p \sqcup p = p$
   - $p \sqcap p = p$

2. **Distributivity of composition over unions and intersections:**
   - $(p_1 \sqcup p_2); p_3 = (p_1; p_2) \sqcup (p_1; p_3)$
   - $p_1; (p_2 \sqcup p_3) = (p_1; p_2) \sqcup (p_1; p_3)$
   - $(p_1 \sqcap p_2); p_3 \subseteq (p_1; p_3) \sqcap (p_2; p_3)$
   - $p_1; (p_2 \sqcap p_3) \subseteq (p_1; p_2) \sqcap (p_1; p_3)$
The Algebra of Operators
The Algebra of External Operators

Theorem (2)

3. properties of the scoping operator
   - $\Phi :: (\Psi :: p) == (\Phi \land \Psi) :: p$
   - $\Phi :: (p_1 \sqcup p_2) = (\Phi :: p_1) \sqcup (\Phi :: p_2)$
   - $\Phi :: (p_1 \sqcap p_2) = (\Phi :: p_1) \sqcap (\Phi :: p_2)$
   - $\Phi :: (p_1; p_2) = (\Phi :: p_1); p_2$
   - $\Phi :: (p_1 \Box p_2) = (\Phi :: p_1) \Box (\Phi :: p_2)$

4. Properties of the provisioning operator
   - $(p \parallel \Phi) \parallel \Psi = p \parallel (\Phi \land \Psi)$
   - $(p_1 \sqcup p_2) \parallel \Phi = (p_1 \parallel \Phi) \sqcup (p_2 \parallel \Phi)$
   - $(p_1 \sqcap p_2) \parallel \Phi = (p_1 \parallel \Phi) \sqcap (p_2 \parallel \Phi)$
   - $(p_1; p_2) \parallel \Phi = p_1; (p_2 \parallel \Phi)$
   - $(p_1 \Box p_2) \parallel \Phi = (p_1 \parallel \Phi) \Box (p_2 \parallel \Phi)$
In summary, theorem 2 shows how policy operators interact with each other.

It also shows that scoping and provisioning operators distribute over unions, intersections, and sequencing operators. So, they can be used to express composed policies.
Internal operators do not satisfy $p \cup p = p$ and $p \cap p = p$, so they do not form a Boolean algebra.

There are some properties for internal operators which are hold for all policies.
Determinism  is closed under some (and not all) internal and external operators.
Consistency

- Contradictory Permission Sets
- Policies Consistence for Authorizations
- Policies Consistent with respect to an Interpretation
- Consistent Policy

Consistency is closed under some (and not all) internal and external operators.
Determinism, Consistency, and Completeness

Completeness

1. Complete Permission Sets
2. Policies Complete for Authorizations
3. Policies Complete with respect to an Interpretation
4. Complete Policy

Completeness is closed under some (and not all) internal and external operators.
Advantages of using logics in security
[Clean Foundation, Flexibility, Declarativeness, Ability of Verification, Independency from Implementation.]

A sample of logic-based policy specification: A calculus for access control

A sample of algebraic approach in policy composition: A propositional algebra for access control
References


References


Thanks for your attention …

Questions?