Last Time (I)

Lecture

- Tools
Outline of Today

Motivations
Outline of Today

Motivations

DAC
Outline of Today

Motivations

DAC

MAC
Outline of Today

Motivations

DAC

MAC

Acces Control Matrix Model
Outline of Today

Motivations
DAC
MAC
Acces Control Matrix Model
RBAC
Outline of Today

Motivations

DAC

MAC

Access Control Matrix Model

RBAC

Non-Interference
  Motivation
  Approach using Type System
Outline of Today

Motivations

DAC

MAC

Access Control Matrix Model

RBAC

Non-Interference
  
    Motivation
  
    Approach using Type System

Conclusion
Outline

Motivations

DAC

MAC

Access Control Matrix Model

RBAC

Non-Interference

Motivation

Approach using Type System

Conclusion
Security policy for university computing

Students

- Access to their own information
- No access to other students’ information unless explicitly given.
- May access and execute a pre-defined selection of files and/or applications. ...
Motivations

Security policy for university computing

Students

- Access to their own information
- No access to other students’ information unless explicitly given.
- May access and execute a pre-defined selection of files and/or applications. ...

POLICY is access restrictions between subjects and objects.
Security policy for e-banking

Bank Customer

- List his account balances
- View his recent transactions.
- Transfer funds from his accounts under conditions
- Get cash at ATM...
Security policy for e-banking

Bank Customer

- List his account balances
- View his recent transactions.
- Transfer funds from his accounts under conditions
- Get cash at ATM...

POLICY is restrictions on objects representing data and processes.
Questions

- How do we formalize such policies?
- What mechanisms can we use to enforce them?
- Can we generalize and abstract?
Motivations

Identity and AAA

**Identification**: associating an identity with a subject.

**Authentication**: verifying the validity of something claimed by a system entity (default assumption: the identity).

**Authorization**: a right or a permission that is granted to a system entity to access a system resource.

**Access Control**: Protection of system resources against unauthorized access.

**Independent Concepts**.

**Exercice**

Give examples of Authentication without Identification.
Give examples of Authorization without Authentication.
Authorization and Access Control

Authorization:

- Specifying who may do what.
- Controlling how these permissions may change.
- Determining which rights subjects have on objects.
- Using matrices, lattices, or other mathematical structures.
Motivations

Mathematical

Structure constitutes state:

\[ S \times O \times R \]

- Changes constitute transitions between states.
- Access rights may depend on the environment.
- Entail obligations for the future.
- Also concerned with enforcement mechanisms.
Example of AAA via centralized reference monitor
(Mandatory Access Control)

System designed so that access requests pass through a gatekeeper. Other components include those for setup, auditing, and recovery.
Motivations

Two main models: DAC & MAC

**Discretionary Access Control**

- Principle: users own resources and control their access.
- Discretionary because subjects may transfer their access rights.
- Example: Unix file system

**Mandatory Access Control**

- Principle: system owner control users access to the resources.
- Mandatory because subjects may not transfer their access rights.
- Example: Military Top Secret information.

MAC is more rigid than DAC, but also more secure.
Outline

Motivations

DAC

MAC

Acces Control Matrix Model

RBAC

Non-Interference
  Motivation
  Approach using Type System

Conclusion
Discretionary Access Control

- **Principle:** users own resources and control their access.
  - *Owner* may change object’s permissions at his discretion.
  - This allows direct or even transitive delegation of rights.
  - Owners may even be able to transfer ownership to other users.
- **Flexible, but open to mistakes, negligence, or abuse.**
  - Requires that all users understand mechanisms and understand and respect the security policy.
  - No control of information dissemination.
DAC example: Unix

- Subjects are users (plus root)
- Objects are files and directories.
- Relations:
  - Each file has an owner and a group.
  - Operations are: read (r), write (w), execute (x).
  - ACLs limited to 9 bits: rwx for user, group, and others.

-rw-r--r-- 1 plafourc users 4158 2007-11-09 12:37 Intro.tex
drwxr-xr-x 5 plafourc users 4096 2007-11-02 14:07 Tools/
DAC example: Unix

Discretionary AC: only file’s owner (and root) can change its ACL.

Direct delegation of rights (rwx) to group or others

- Unix supports additional right delegation
  - using setuid [or setgid].
  - Executor takes on owner’s user [group] identity during execution.
  - Example: users automatically “upgraded” to root to change their passwords in the password file.
DAC example: Limitations

- Open to abuse and source of many security holes.
- Not all policies can be directly mapped onto this mechanism.

Question

How would we express that a patient can read but not write his medical records at a hospital?
DAC example: Limitations

- Open to abuse and source of many security holes.
- Not all policies can be directly mapped onto this mechanism.

Question

How would we express that a patient can read but not write his medical records at a hospital? Who owns the records?
Outline

Motivations

DAC

MAC

Acces Control Matrix Model

RBAC

Non-Interference
  Motivation
  Approach using Type System

Conclusion
Mandatory Access Control

AC decisions compare

- security labels: indicating sensitivity/criticality of objects.
- formal authorization: i.e. security clearances, of subjects.

MAC policies often identified with multilevel security policies.
Mandatory Access Control

Specifies system-wide access restriction to objects.

- **Mandatory** because subjects may not transfer their access rights.
- Shifts power from users to system owner.
- Required by US DOD for developing “trusted systems”.
- More rigid than DAC, but also more secure.
Formalism (+ terminology) comes from US DOD “Orange Book”.

Formalism combines
- a linearly ordered ranks or sensitivity levels, and
- a lattice of compartments.
Labels (cont.)

- Labels combine classifications on personal and data.
  \[ \text{Class} = \langle \text{rank, compartment} \rangle \]

- Dominance relation defined component-wise.
  \[ (r_1, c_1) \leq (r_2, c_2) :\Leftrightarrow r_1 \leq r_2 \land c_1 \subseteq c_2 \]

  Example: \( (\text{secret, Iraq}) \leq (\text{top secret, Middle East}) \)

Authorization based on comparing labels.
Lattices

Definition

A lattice \((L, \leq)\) consists of a set of \(L\) and a partial ordering \(\leq\), so that for every 2 elements \(a, b \in L\) there exists a least upper bound \(u \in L\) and a greatest lower bound \(l \in L\), i.e.

\[
a \leq u, \quad b \leq u \quad \text{and} \quad (a \leq v \& b \leq v) \rightarrow (u \leq v) \text{ for all } v \in L
\]

\[
l \leq a, \quad l \leq b \quad \text{and} \quad (k \leq a \& k \leq b) \rightarrow (k \leq l) \text{ for all } k \in L
\]
Lattices

**Definition**

A lattice \((L, \leq)\) consists of a set of \(L\) and a partial ordering \(\leq\), so that for every 2 elements \(a, b \in L\) there exists a least upper bound \(u \in L\) and a greatest lower bound \(l \in L\), i.e.

\[
\begin{align*}
\text{If } a &\leq u, \ b \leq u \quad \text{and} \quad (a \leq v \ &\& \ b \leq v) \rightarrow (u \leq v) \quad \text{for all } v \in L \\
\text{If } l \leq a, \ l \leq b \quad \text{and} \quad (k \leq a \ &\& \ k \leq b) \rightarrow (k \leq l) \quad \text{for all } k \in L
\end{align*}
\]

If security labels form a lattice, we can uniquely answer questions:

- Given two objects with different labels, what is the minimal label a subject requires to be allowed to read both objects?
- Given two subjects with different labels, what is the maximal label an object can have that can still be read by both subjects?
The Bell-LaPadula (BLP) Model (1975)

Models security policies for confidentiality.

- Authorization for subjects reading and writing objects.
- Combines state-transition systems with partial-orders on labels.

**Access decisions**

**No Read-Up** (also called Simple Security Property).
A subject with label $x_s$ can only read information in an object with label $x_o$ if $x_s$ dominates $x_o$.

**No Write-Down** (also called *-Property).
A subject with label $x_s$ can only write information to an object with security label $x_o$ if $x_o$ dominates $x_s$.

⇒ You may only read below your classification and write above it.
BLP — (dis)allowed operations

Write

Read

Write

Write

Read
BLP (cont.)

- Labels cannot be changed. No information leakage possible!
- But also prevents “legitimate” communication from high-level subjects to low-level ones!!

Possible solutions:
BLP (cont.)

- Labels cannot be changed. No information leakage possible!
- But also prevents “legitimate” communication from high-level subjects to low-level ones !!

Possible solutions:
  - Temporarily downgrade the subject’s security level.
  - Identify a set of trusted subjects that may violate the \( \ast \)-property.

- Mechanism support provided by some systems, e.g., BLP module for NSA’s SELinux.
Biba Integrity Model  (1977)

- Dual to BLP
  - No Write-up: The writer’s label must dominate the object’s.
  - No Read-down: The object’s label must dominate the reader’s.
  - ⇒ only write below your classification and read above it.

- But what if you want both confidentiality and integrity?
  - Use BLP for classifying some data, and Biba for others.
  - Alternatively, only read and write at same classification.
Limitations of Access Control Models

- AC models restrict operations like read and write. But information may be revealed in other ways.
- Examples of information leaks.
  - Error messages to user, e.g., “file not found”.
  - CPU usage: timing behavior, power consumption, noise, ...
  - Locking and unlocking files.
  - Sending and delaying messages.
  - Encode information in the invoice sent for services rendered.
Outline

Motivations

DAC

MAC

Access Control Matrix Model

RBAC

Non-Interference
  Motivation
  Approach using Type System

Conclusion
Access Control Matrix Model

- Simple framework for describing a protection system by describing the privileges of subjects on objects.
  
  Subjects: users, processes, agents, groups, ...
  
  Objects: data, memory banks, other processes, ...
  
  Privileges (or permissions/rights): read, write, modify, ...

- A reference monitor decides on requests.

- Constitutes a model and, when implemented, a mechanism.
Protection state

- A protection state (relative to a set of privileges $P$) is a triple $(S, O, M)$.
  - A set of current subjects $S$.
  - A set of current objects $O$.
  - A matrix $M$ defining the privileges for each $(s, o) \in S \times O$, i.e., a relation $S \times O \times P$ or equivalently a function $S \times O \rightarrow \mathcal{P}(P)$.

Example

<table>
<thead>
<tr>
<th>Subjects</th>
<th>File 1</th>
<th>File 2</th>
<th>File 3</th>
<th>File 4</th>
<th>Account 1</th>
<th>Account 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Own R</td>
<td>W</td>
<td></td>
<td>Inquiry Credit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>R</td>
<td>Own R</td>
<td>W</td>
<td>R</td>
<td>Inquiry Debit</td>
<td>Inquiry Credit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charlie</td>
<td>R</td>
<td>W</td>
<td>Own R</td>
<td>Inquiry Debit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>X</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Transitions

- **State transitions** are modeled by a set of commands.

- **Commands** expressed in terms of 6 primitive operations:
  1. **enter** \( p \) into \( M(s, o) \) (for \( p \in P \))
  2. **delete** \( p \) from \( M(s, o) \) (for \( p \in P \))
  3. **create** subject \( s \)
  4. **destroy** subject \( s \)
  5. **create** object \( o \)
  6. **destroy** object \( o \)

- **Operation semantics** as expected, e.g., **enter** \( p \) into \( M(s, o) \).

  **Precondition:** \( s \in S \) and \( o \in O \)

  **New state:** \( S' = S, \ O' = O, \ M'(s, o) = M(s, o) \cup \{p\}, \)
  and \( M'(s_i, o_i) = M(s_i, o_i), \) for \( (s'_i, o'_i) \neq (s, o) \)
Examples of commands

- Consider a system where users own files and can delegate permissions directly (e.g., ConferRead) or transitively (e.g., DelegateRead) for reading and writing them.
- Privileges: ownership (own), read (R), and write (W).
- Commands might include:

```plaintext
command CreateFile(s, f)
  create object f
  enter Own into M(s, f)
  enter R into M(s, f)
  enter W into M(s, f)
end
```
Transition system semantics

- Write \((S, O, M) \vdash_c (S', O', M')\) to denote a transition associated with the command \(c\). For example:

\[
\begin{array}{c|c|c}
\text{Alice} & \text{Own} & \text{File 1} \\
\hline
\text{Own} & R & \\
\text{Bob} & R & \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{Alice} & \text{Own} & \text{File 1} \\
\hline
\text{Own} & R & \\
\text{Bob} & R & \\
\hline
\end{array}
\]

- A starting state \(st_0 = (S_0, O_0, M_0)\) and a set of commands \(C\) determines a state-transition system.
- So a model describes a set of system traces, namely those traces

\[st_0, st_1, st_2, st_3, \ldots\]

where \(st_i \vdash_{c_i} st_{i+1}\), for \(c_i \in C\).
Access matrix: data structures

Matrices define access rights and provide a basis for different possible enforcement mechanisms.

**Access Matrix**  AC List (ACL)  Capabilities List

Represent as 2 dimensional objects or set of 1-dimensional objects.
Access-control (authorization) list

- **ACL**: use lists to express view of each object $o$:
  
  $i$th entry in the list gives the name of a subject $s_i$ and the rights $r_i$ in $M(s_i, o)$ of the access-matrix.

- Standard example: AC for files.
Access-control lists (cont.)

- Implementation:
  - Associate ACL with each object, typically maintained by OS, middleware, server, ...
  - Check user (group, ...) against list.
  - Relies on authentication: need to know user.

- Usually used for discretionary access control.
  Owners have the (usually sole) authority to grant or revoke rights to the objects they own to other users.

- ACLs are found, for example in the DEC VMS operating system, Linux, and Windows NT.
Capability list

- Subject view of AC matrix.
  A capability is essentially a pair: an object and an operation.

- Users should not be able to forge capabilities.
  Centralized systems:
  OS manages capabilities in protected address space.

  Distributed systems:
  - Pair protected using cryptography, e.g., signatures.
  - Reference monitor checks ticket.
  - Need not know identity of user or process (at least if transitive delegation is allowed).

- Capabilities not often used, but gaining popularity in distributed (e.g., mobile agent) setting.
ACLs versus capabilities

ACLs

- ACLs are compact and easy to review.
- Deleting an object is simple.
- Deleting a subject more difficult.
- Delegation possible in discretionary access control setting: Owners have the (usually sole) authority to grant or revoke rights to the objects they own to other users.

Capabilities

(in particular, when distributed)

- Not so compatible with object-oriented view of the world.
- Delegation easy, revocation difficult.
- In general, difficult to know who has permissions on an object.
Outline

Motivations

DAC

MAC

Access Control Matrix Model

RBAC

Non-Interference
  Motivation
  Approach using Type System

Conclusion
Scalable security policies

- How do we formalize a policy when there are $10^3 - 10^6$ subjects and objects? An example?
Scalable security policies

- How do we formalize a policy when there are $10^3 - 10^6$ subjects and objects? An example? Your typical bank!
  - AC matrices (whether ACLs or CLs) scale poorly.
  - They are difficult (or impossible) to maintain.
- Overcome using standard tricks: abstraction and hierarchy.
  Abstraction: Many subjects (or objects) have identical attributes, and policy is based on these attributes.
  Hierarchy: Often functional/organizational hierarchies that determine access rights.
First, a slight reformulation
to set the stage for Role-Based Access Control

- Recall AC-Matrix: $M$ defines a relation $S \times O \times P$, where $P$ is a privilege like “read” or “write”.
- We now recast matrix $M$ as relation, $M \subseteq \text{Users} \times \text{Permissions}$
- A permission represents authorization to perform an operation on an object.
  In matrix-model terminology: a pair (object, privilege) $\in O \times P$.
- Declarative access control: authorization specified by a relation.
  A user is granted access iff he has the required permission.
  $u \in \text{Users} \text{ has } p \in \text{Permissions} : \iff (u, p) \in \text{AC}$. 
## Access Control — A Simple Example

<table>
<thead>
<tr>
<th>User</th>
<th>User</th>
<th>Permission</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Alice</td>
<td>read file a</td>
<td>read file a</td>
</tr>
<tr>
<td>Alice</td>
<td>Alice</td>
<td>write file a</td>
<td>write file a</td>
</tr>
<tr>
<td>Alice</td>
<td>Bob</td>
<td>start application (x)</td>
<td>start application (x)</td>
</tr>
<tr>
<td>Alice</td>
<td>Bob</td>
<td>start application (y)</td>
<td>start application (y)</td>
</tr>
<tr>
<td>Bob</td>
<td>Bob</td>
<td>read file a</td>
<td>write file a</td>
</tr>
<tr>
<td>John</td>
<td>John</td>
<td>write file a</td>
<td>start application (x)</td>
</tr>
<tr>
<td>John</td>
<td>John</td>
<td>start application (x)</td>
<td>read file a</td>
</tr>
<tr>
<td>John</td>
<td>John</td>
<td>start application (x)</td>
<td>write file a</td>
</tr>
</tbody>
</table>
Role-Based Access Control

- **Role-Based Access Control** decouples users and permissions by introducing roles.
- Formalized by a set Roles and the relations $UA \subseteq \text{Users} \times \text{Roles}$ and $PA \subseteq \text{Roles} \times \text{Permissions}$, where

$$AC := PA \circ UA$$

$$AC := \{(u, p) \in \text{Users} \times \text{Permissions} | \exists r \in \text{Roles} : (u, r) \in UA \land (r, p) \in PA\}.$$

- **Example**:

<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>User</td>
</tr>
<tr>
<td>Alice</td>
<td>Superuser</td>
</tr>
<tr>
<td>Bob</td>
<td>User</td>
</tr>
<tr>
<td>John</td>
<td>User</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Role</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>read file a</td>
</tr>
<tr>
<td>User</td>
<td>write file a</td>
</tr>
<tr>
<td>User</td>
<td>start application x</td>
</tr>
<tr>
<td>Superuser</td>
<td>start application y</td>
</tr>
</tbody>
</table>
RBAC — advantages over matrix

- Roles are abstraction of jobs or functions in an organization.
  - Distinct from notion of user groups, which names collections of users.
  - Emphasis is on responsibility and associated permissions.
- Increases abstraction in policies. Policies become more manageable.
- Less information must usually be maintained when number of roles is small (relative to number of users and permissions).

\[ |PA| + |UA| \leq |AC| \]
RBAC — Extensions

1. Factorization idea can be further extended by introducing a partial order $\geq$ on roles.

$$\text{AC} := \text{PA} \circ \geq \circ \text{UA}$$

Semantics: larger roles inherit permissions from all smaller roles.
E.g., Cardiologist $\geq$ Physician, so cardiologists have physicians’ rights.

2. Hierarchies on users (UA) and permissions (PA) also possible.

3. RBAC standard introduces additional extensions.
E.g., introduces notion of sessions, representing users’ active roles.
An example

E.g., User 9 can carry out operation 2.
Outline

Motivations

DAC

MAC

Acces Control Matrix Model

RBAC

Non-Interference
  Motivation
  Approach using Type System

Conclusion
Motivation

Malicious and/or buggy code is a threat:

- code from untrusted source (applet, javascript, ...)
- code interacts with your confidential data and with the outside world

Problem: does program $p$ leak information?

Information flow analysis: Cohen, Denning in the 70’s.

Non-interference (Goguen & Meseguer): semantic definition of absence of information leakage.
Non-interference

The basic scenario:

- Public variables $\ell, \ell_1, \ldots$
- Secret variables $h, h_1, h_2, \ldots$

Explicit leak

Program $\ell := h_1$ leaks: the value of $h$ is copied into $\ell$. 
Non-interference

The basic scenario:

- Public variables $\ell, \ell_1, \ldots$
- Secret variables $h, h_1, h_2, \ldots$

**Explicit leak**

Program $\ell := h_1$ leaks: the value of $h$ is copied into $\ell$.

**Implicit leak**

Program $\textbf{if } h_1 = h_2 \textbf{ then } \ell := 0 \textbf{ else } \ell := 1 \textbf{ fi}$ leaks: the final value of $\ell$ depends on the value of $h_1$ and $h_2$. 
Non-interference

The formal definition of *Non-interference* depends on the type of the considered system:

- Deterministic.
- Non-deterministic.
- Probabilistic.
- Cryptographic.
Non-interference definitions

Henceforth, let $L$ (resp. $H$) be a set of low (resp. high) variables.

Deterministic case.

The denotation of a program is a mapping $f : \Sigma \rightarrow \Sigma$. Then, $f$ is non-interfering, if

$$\forall S, S' \in \Sigma \cdot S_L = S'_L \Rightarrow f(S)_L = f(S')_L$$

Non-deterministic case (Possibilistic non-interference).

The denotation of a program is a mapping $f : \Sigma \rightarrow 2^\Sigma$. Now, $f$ is said non-interfering, if

$$\forall S, S' \in \Sigma \cdot S_L = S'_L \Rightarrow f(S)_L = f(S')_L$$

where

i.e. $\forall S_1 \in f(S) \exists S_2 \in f(S') \cdot S_1_L = S_2_L$ and vice versa.
Non-interference definitions (cntd.)

Probabilistic case.

Let $\mathcal{D}(\Sigma)$ be the set of distributions on $\Sigma$. The denotation of a program is a mapping $f : \mathcal{D}(\Sigma) \rightarrow \mathcal{D}(\Sigma)$.

For a distribution $D \in \mathcal{D}(\Sigma)$, define $D_L \in \mathcal{D}(\Sigma_L)$ with

$$D_L(S_L) = \sum_{S' \in \Sigma, S'_L = S_L} D(S').$$

$f$ is non-interfering, if

$$\forall D, D' \in \mathcal{D}(\Sigma) \cdot D_L = D'_L \Rightarrow f(D)_L = f(D')_L$$
Example

Possibilistic NI does not imply probabilistic NI:

\[(\ell := (h_1 = h_2)) \oplus \frac{1}{2} (\ell := 0 \oplus \frac{1}{2} \ell := 1)\]
Motivation

Example

Possibilistic NI does not imply probabilistic NI:

\[
(\ell := (h_1 = h_2)) \oplus \frac{1}{2} (\ell := 0 \oplus \frac{1}{2} \ell := 1)
\]

Possibilistic NI

- \( h_1 = h_2 \) then \( \ell = 0 \) and \( \ell = 1 \)
- \( h_1 \neq h_2 \) then \( \ell = 0 \) and \( \ell = 1 \)
Example

Possibilistic NI does not imply probabilistic NI:

$$(\ell := (h_1 = h_2)) \oplus \frac{1}{2} (\ell := 0 \oplus \frac{1}{2} \ell := 1)$$

Possibilistic NI

- $h_1 = h_2$ then $\ell = 0$ and $\ell = 1$
- $h_1 \neq h_2$ then $\ell = 0$ and $\ell = 1$

No Probabilistic NI

- $h_1 = h_2$ then $P[\ell = 0] = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$ and $P[\ell = 1] = \frac{1}{4}$
- $h_1 \neq h_2$ then $P[\ell = 0] = \frac{1}{4}$ and $P[\ell = 1] = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$
Non-Interference

Cryptographic non-interference

The denotation of a program is a function $f : \Sigma \rightarrow \mathcal{D}(\Sigma)$ which can be canonically lifted to a function $f : \mathcal{D}(\Sigma) \rightarrow \mathcal{D}(\Sigma)$.

1. if $\mathcal{D}_L \sim \mathcal{D}'_L$ then $f(\mathcal{D})_L \sim f(\mathcal{D}')_L$.
2. $[x \leftarrow \mathcal{D} : (f(x)_L, x_H)] \sim [x, y \leftarrow \mathcal{D} : (f(x)_L, y_H)]$ [Laud]
3. $| \Pr[f(S[h := 1])(\ell) = 1] - \Pr[f(S[h := 0])(\ell) = 1] |$ is negligible [SA’06].
5. Possibilistic-symbolic [AHS’06]:
   A symbolic semantics à la Dolev-Yao combined with the Possibilistic non-interference notion:
   \[
   \ell := \text{Enc}(k, 0); \text{if } h \text{ then } \ell_1 := \text{Enc}(k, 0) \text{ else } \ell_1 := \ell \text{ fi}
   \]
   \[\Rightarrow\] distinguish encryption occurrences.
Motivation

Problem statement

Our aim is to automatically check cryptographic non-interference of programs that use random assignments and deterministic encryption: if $D_L \sim D'_L$ then $f(D)_L \sim f(D')_L$

Methods to ensure Non-Interference

- At Compile time:
  - Static Analysis
  - Type Checking (Volpano & Smith)

- At run-time: monitoring
Non-Interference

Motivation

Example

Simple programs

\[ T[3] = h; \ l = T[3]; \]
Example

Simple programs

\[ T[3] = h; l = T[3]; \text{leaks} \]
Non-Interference

Motivation

Example

Simple programs

\[ T[3] = h; \ l = T[3]; \ \text{leaks} \]
\[ T[3] = l; \ h = T[3]; \]
Example

Simple programs

T[3] = h; l = T[3]; leaks
T[3] = l; h = T[3]; Does not leak

With a loop

for (i=0; i<T.length; i++) T[i]=0;
T[h]=1;
Motivation

Example

Simple programs

\[ T[3] = h; l = T[3]; \text{leaks} \]
\[ T[3] = l; h = T[3]; \text{Does not leak} \]

With a loop

\[
\text{for (i=0; i<T.length; i++) T[i]=0;  } \\
T[h]=1; \text{ leaks information }
\]
Motivating Example (cf. [Volpano’00])

Encryption does not guarantee absence of leakage:

```
l := 0^\eta; m := 0^{\eta-1}1;
for i := \eta to 1 do
\begin{align*}
\ell_1 & := \text{Enc}(k, h|m); \\
\ell_2 & := \text{Enc}(k, h);
\end{align*}
\text{if } (\ell_1 = \ell_2) \text{ then } l := l|m \text{ else skip fi ;}
\begin{align*}
m & := m \ll 1 \od
\end{align*}
```

$m, \ell, \ell_1, \ell_2$ low-security variables,
$k, h$ high-security variables,
parameter $\eta$ size of blocks
Example-cntd.

\[ l := 0^n; m := 0^{n-1}11; \]
\[ \text{for } i := \eta \text{ to } 1 \text{ do } \]
\[ l_1 := \text{Enc}(k, \nu l_r \cdot (h|m) + l_r); \]
\[ l_2 := \text{Enc}(k, h + l_r); \]
\[ \text{if } (l_1 = l_2) \text{ then } l := l|m \text{ else skip fi}; \]
\[ m := m \ll 1 \text{ od} \]
Non-Interference

Motivation

Example cntd

\[
\begin{align*}
l &:= 0^n; m := 0^{n-1}1; \\
\text{for } i := \eta \text{ to } 1 \text{ do} & \quad l_1 := \text{Enc}(k, \nu l_r \cdot (h|m) + l_r); \\
& \quad l_2 := \text{Enc}(k, \nu l_r \cdot h + l_r); \\
& \quad \text{if } (l_1 = l_2) \text{ then } l := l|m \text{ else skip fi }; \\
& \quad m := m \ll 1 \text{ od}
\end{align*}
\]
Objectives

- Develop a type system for non-interference for programs that use deterministic encryption
- Establish soundness in the exact (concrete) security model

A type system for a simple imperative language

Expressions:
\[(x, \tau) \in \Gamma \]
\[\Gamma \vdash x : \tau\]

Sub-typing: \( L \subseteq H \)
\[\Gamma \vdash S : \tau' \quad \tau \subseteq \tau' \]
\[\Gamma \vdash S : \tau\]

Commands:
\[(x, \tau) \in \Gamma \quad \Gamma \vdash e : \tau\]
\[\Gamma \vdash x := e\]
\[\Gamma \vdash S_1 : \tau \quad \Gamma \vdash S_2 : \tau\]
\[\Gamma \vdash S_1; S_2 : \tau\]
\[\Gamma \vdash e : \tau \quad \Gamma \vdash S_1 : \tau \quad \Gamma \vdash S_2 : \tau\]
\[\Gamma \vdash \text{if } e \text{ then } S_1 \text{ else } S_2 \text{ fi} : \tau\]
\[\Gamma \vdash e : \tau \quad \Gamma \vdash S : \tau\]
\[\Gamma \vdash \text{while } e \text{ do } S \text{ od} : \tau\]
Using the same type system but something more is needed.

Sound but what about $\ell_1 := h_1 \oplus \text{random()}$?

- Ill-typed for usual type systems.
- Does not leak information (equivalent to $\ell_1 := \text{random()}$).
- We want to accept such a program (one-time pad).
Non-Interference

Approach using Type System

Quizz

Should we typecheck/What kind of type should we introduce to typecheck

- $\ell_1 := \nu h_2 \cdot h_2 \oplus h_1$?
- $\ell_1 := \nu \ell_2 \cdot \ell_2 \oplus h_1$?
- $\ell_1 := \pi(h_1)$? ($\pi$ random permutation)
- $\ell_1 := \pi(\nu \ell_2 \cdot \ell_2 \oplus h_1)$?

Types: $(\tau, \theta)$ where
- $\tau \in \{L, H\}$
- $\theta \in \{L^r, H^r, \top\}$
Type systems for non-interference

- Introduce type $L$ for low-sensitive data/variables and $H$ high-sensitive data/variables.
- Arithmetic operator/assignment apply to and return data of the same type.
- Subtyping: expressions in $L$ also belong to $H$.

Examples:

$$h_1 := h_2 \quad \frac{(x, \tau) \in \Gamma}{\Gamma \vdash e : \tau} \quad \frac{\Gamma \vdash x := e : \tau}{\Gamma \vdash e_1 : \tau} \quad \frac{\tau \sqsubseteq \tau'}{\Gamma \vdash e_2 : \tau'} \quad L \sqsubseteq H$$

$$h_1 := \ell_1 \quad \Gamma \vdash e : \tau \quad \ell_1 := h_1 \oplus h_2$$

$$\ell_1 := h_1 \oplus h_2$$
Non-interference with random permutation

Do these programs leak:

1. $\ell_1 := \pi(h_1 \oplus \nu \ell_2 \cdot \ell_2)$
2. $\ell_1 := \pi(h_1 \oplus \nu \ell_2 \cdot \ell_2); \ell_3 := \pi(\ell_2)$

Problem: once used to randomize argument of a permutation, $\ell_2$ must not appear under a permutation.

$\rightarrow$ We need to introduce a set of forbidden variables (not reuse random variable in $\pi$), and add this set to the premises of our typing judgments.
Non-Interference

Approach using Type System

An example CBC

\[ \nu l_0; \]
\[ l_1 := \text{Enc}(k, l_0 \oplus h_1); \ldots; l_i := \text{Enc}(k, l_{i-1} \oplus h_i), \ldots, \]
\[ l_n := \text{Enc}(k, l_{n-1} \oplus h_n) \]

- \( l_0 \) is typed \((L, L^r)\).
- \( l_i \oplus h_{i+1} \) is typed \((H, L^r)\).
- \( \text{Enc}(k, l_i \oplus h_i) \) is typed \((L, L^r)\).
- \( l_i \) is added to \( G \).
Type system for non-interference of randomized programs

New typing rules for expressions:

\[
\Gamma \vdash e_1 : (\tau_1, \theta_1) \quad \Gamma \vdash e_2 : (\tau_2, \theta_2) \\
\Gamma \vdash e_1 \oplus e_2 : (\tau_1 \sqcup \tau_2, \theta_1 \cap \theta_1)
\]

\[
\Gamma(x) = \tau \\
\Gamma \vdash \nu x \cdot x : (\tau, \tau')
\]

\[
\Gamma(x) = \tau \\
\Gamma \vdash x : (\tau, \top)
\]

For programs without random permutation

Soundness result: well-typedness implies non-interference.
Proof of non-interference for random permutation

Result: soundness of typing for random permutation.

The result

For any type environment $\Gamma$, command $c$ and pairs of distribution ensembles in $\mathcal{D}(\Sigma)$ such that

- $\Gamma, \emptyset, \emptyset \vdash c : \tau$, for some type $\tau \in \{H, L\}$, and
- $X|_L \sim_{(t, \epsilon)} Y|_L$.

we have

$$\llbracket c \rrbracket(X)|_L \sim (t - \mathcal{T}(c), \epsilon') \llbracket c \rrbracket(Y)|_L \text{ where } \epsilon' = \frac{2\mathcal{T}(c)^2}{|U|} + \epsilon.$$

- $\mathcal{T}(c)$ the number of calls of $\pi$
- $\mathcal{I}(c)$ the running time of $c$
Main ideas of the proof

Ideas:

- Reduction to programs without permutation
- Distinguishing occurrences of permutation applied to random expressions ($\pi^r$) and to low security expressions ($\pi^\top$)
- Simulate $\pi^\top$ by an unknown function and $\pi^r(e)$ by random sampling (independent of its argument).

Simulation might fail, but with quantifiable low risk.

$$[[c^p](X)]_L \sim_{\text{sim}} [[c](X)]_L \sim_{1st} [[c](Y)]_L \sim_{\text{sim}} [[c^p](Y)]_L$$
Non-interference with encryption

Now, we replace the permutation $\pi$ by encryption.

**Cipher**

A cipher block is a *family of permutations* $\Pi : \text{Keys}(\Pi) \times \mathcal{D} \rightarrow \mathcal{D}$, where $\text{Keys}(\Pi)$ is the key space of $\Pi$, and for any $k \in \text{Keys}(\Pi)$, $\Pi(k, \cdot)$ is a permutation onto $\mathcal{U}$. 
Non-Interference
Approach using Type System

Security assumption

The usual security notion for ciphers is Pseudo-randomness.

Experiment \( \text{PRP}_b^{\eta}(A) \):
\[
\begin{align*}
 k &\leftarrow \text{Keys}(\Pi); \quad P \leftarrow \text{SP} ; \\
 O_0 &= P; \quad O_1 = \text{Enc}(k, \cdot); \\
 b' &\leftarrow A^{O_b}()
\end{align*}
\]

The PRP advantage of \( A \) is defined as
\[
\text{Adv}_{\Pi}^{\text{prp}}(A) = \Pr[\text{PRP}_{1}^{\eta}(A) = 1] - \Pr[\text{PRP}_{0}^{\eta}(A) = 1].
\]

An encryption scheme \( \Pi \) is said to be a \((t, \epsilon)\)-pseudo-random permutation, denoted \((t, \epsilon)\)-PRP, if for any PRP adversary \( A \), that runs in time \( t \), \( \text{Adv}_{\Pi}^{\text{prp}}(A) \leq \epsilon \).
Outline

Motivations
DAC
MAC
Acces Control Matrix Model
RBAC
Non-Interference
  - Motivation
  - Approach using Type System

Conclusion
Summary

Today

- Access Control
- Matrix Model
- MAC
- DAC
Next Time

- Side Channels
- Link between Cryptographic and Symbolic World
Thank you for your attention

Questions ?