

# Advanced Cryptography 1st Semester 2007-2008

## Acces Control

Pascal Lafourcade

*Université Joseph Fourier, Verimag*

Master: November 26th 2007 thanks to David Basin and  
Yassine Lahknech

## Last Time (I)

### Lecture

- Tools

# Outline of Today

- 1 Motivations
- 2 Access Control Matrix Model
- 3 RBAC
- 4 DAC
- 5 MAC
- 6 Non-Interference
- 7 Conclusion

# Outline

- 1 Motivations
- 2 Access Control Matrix Model
- 3 RBAC
- 4 DAC
- 5 MAC
- 6 Non-Interference
- 7 Conclusion

## Two examples

- Security policy for university computing  
A student has full access to information that she created. Students have no access to other students' information unless explicitly given. Students may access and execute a pre-defined selection of files and/or applications. ...  
Describes **access restrictions** between **subjects** and **objects**.
- Security policy for e-banking  
A bank customer may list his account balances and recent transactions. He may transfer funds from his accounts provided his total overdraws are under 10,000 Euros. Transfers resulting in larger overdraws must be approved by his account manager. ...  
Describes **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?
  - Rather than studying **individual** policies and mechanisms, can we define and study **general classes** of them which abstract away from concrete characteristics of specific domains?
  - These classes are called **security models**.
- **Scope limitations:** We will not handle security engineering aspects like requirements analysis, policy refinement, etc.

## Identity and AAA

**Identification:** The process of associating an identity with a subject.

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

**Authorization:** An authorization is 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; a process by which use of system resources is regulated according to a security policy and is permitted by only authorized entities (users, programs, processes, or other systems) according to that policy.

Concepts independent.

### Exercice

Give examples of Authentication without Identification.

Give examples of Authorization without Authentication.

## Authorization and Access Control

- Typical access control models focus on authorization, i.e., specifying who may do what, and controlling how these permissions may change.
- Authorization specified using matrices, lattices, or other mathematical structures, which specify which rights subjects have on objects.
  - Simplest case amounts to a mathematical relation  $S \times O \times R$ .
  - Structure constitutes state. Changes constitute transitions between states.
  - Setup quite complex in practice. Access rights may depend on the environment and even entail obligations for the future
- Access control also is concerned with enforcement mechanisms.

## Two main models: MAC & DAC

### Mandatory Access Control

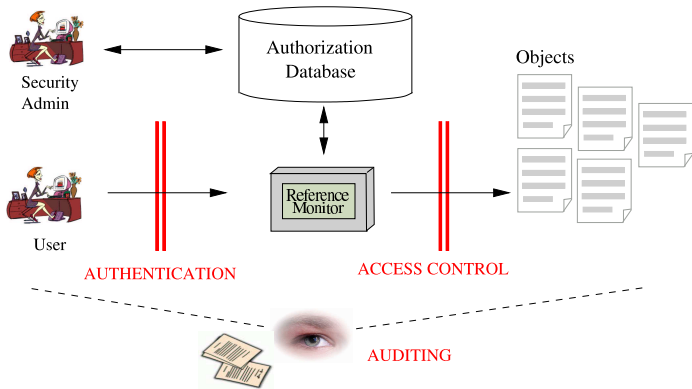
- Principle: system owner control users access to the resources.
- **Mandatory** because subjects may not transfer their access rights.
- Example: Military Top Secret information.

### Discretionary Access Control

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

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

## 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.

# Outline

- 1 Motivations
- 2 Access Control Matrix Model**
- 3 RBAC
- 4 DAC
- 5 MAC
- 6 Non-Interference
- 7 Conclusion

## Access Control Matrix Model

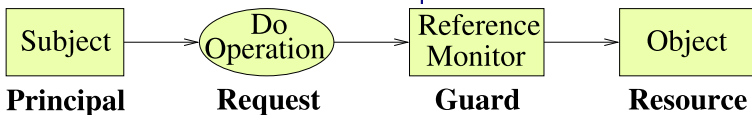
- Simple framework for describing a protection system by describing the **privileges** of **subjects** on **objects**.

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 Objects

		File 1	File 2	File 3	File 4	Account 1	Account 2	
Subjects	Alice	Own R W		W X		Inquiry Credit		.....> Privileges (rights)
	Bob	R	Own R W	W	R	Inquiry Debit	Inquiry Credit	
	Charlie	R W	R		Own R X		Inquiry Debit	

## 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  $(s'_i, o'_i) \neq (s, o)$

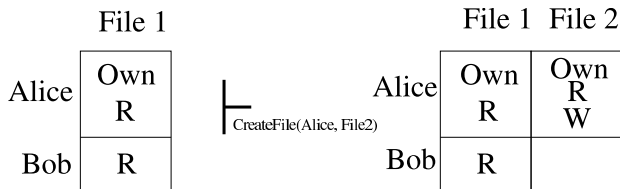
## Examples of commands

- 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:

```
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:



- 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

$$st_0, st_1, st_2, st_3, \dots$$

where  $st_i \vdash_{c_i} st_{i+1}$ , for  $c_i \in C$ .

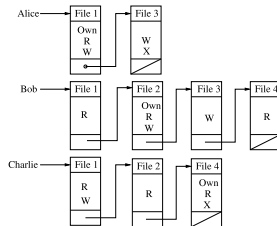
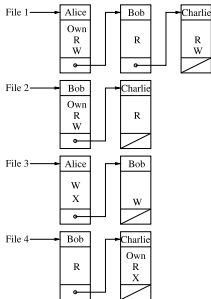
## Access matrix: data structures

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

### Access Matrix AC List (ACL)

### Capabilities List

	File 1	File 2	File 3	File 4	Account 1	Account 2
Alice	Own R W		W X		Inquiry Credit	
Bob	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
Charlie	R W	R		Own R X		Inquiry Debit

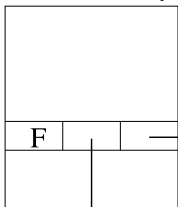


Represent as 2 dimensional objects or set of 1-dimensional objects.

## Access-control (authorization) list

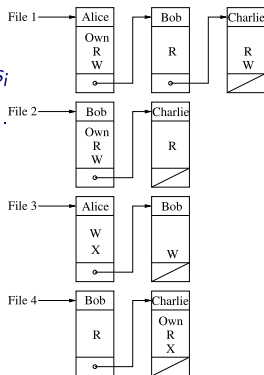
- **ACL**: use lists to express view of each object  $o$ :  
 $i$ th entry in the list gives the name of a subject  $s_i$   
 and the rights  $r_i$  in  $M(s_i, o)$  of the access-matrix.
- Standard example: AC for files.

### File Directory



### ACL for F

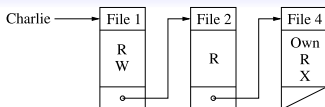
User ID	Rights
Alice	Own, RW
Bob	RW
Charlie	R
John	R



## 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

	File 1	File 2	File 3	File 4	Account 1	Account 2
Alice	Own R W		W X		Inquiry Credit	
Bob	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
Charlie	R W	R		Own R X		Inquiry Debit

## 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

- 1 Motivations
- 2 Access Control Matrix Model
- 3 RBAC**
- 4 DAC
- 5 MAC
- 6 Non-Interference
- 7 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.

$$u \in \text{Users has } p \in \text{Permissions} : \iff (u, p) \in \text{AC}.$$

## Access Control — A Simple Example

User
Alice
Bob
John

User	Permission
Alice	read file a
Alice	write file a
Alice	start application x
Alice	start application y
Bob	read file a
Bob	write file a
Bob	start application x
John	read file a
John	write file a
John	start application x

Permission
read file a
write file a
start application x
start application y

## 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

$$AC := PA \circ UA$$

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

- **Example:**

User	Role
Alice	User
Alice	Superuser
Bob	User
John	User

Role
User
Superuser

Role	Permission
User	read file a
User	write file a
User	start application x
Superuser	start application y

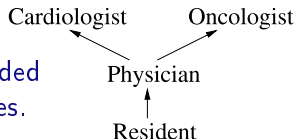
## RBAC — advantages over matrix

- 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.



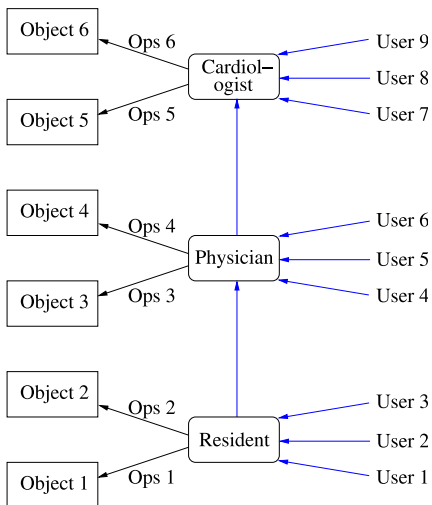
$$AC := PA \circ \geq \circ UA$$

Semantics: larger roles inherit permissions from all smaller roles.

E.g.,  $\text{Cardiologist} \geq \text{Physician}$ , 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

- 1 Motivations
- 2 Access Control Matrix Model
- 3 RBAC
- 4 DAC**
- 5 MAC
- 6 Non-Interference
- 7 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**) and objects are files and directories.
- Each file has an owner and a group.
- Operations are: read (r), write (w), execute (x).  
For directories, r = “list” and x = “usable in constructing paths”.
- ACLs typically 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/
```

- Discretionary AC: only file's owner (and **root**) can change its ACL.  
This allows direct delegation of rights (rwx) to group or others.
- Open to abuse and source of many security holes.
- Not all policies can be directly mapped onto this mechanism.  
How would we express that a patient can read but not write his medical records at a hospital?

## DAC example: Unix

- Subjects are users (plus **root**) and objects are files and directories.
- Each file has an owner and a group.
- Operations are: read (r), write (w), execute (x).  
For directories, r = “list” and x = “usable in constructing paths”.
- ACLs typically 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/
```

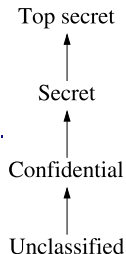
- Discretionary AC: only file's owner (and **root**) can change its ACL.  
This allows direct delegation of rights (rwx) to group or others.
- Open to abuse and source of many security holes.
- Not all policies can be directly mapped onto this mechanism.  
How would we express that a patient can read but not write his medical records at a hospital? Who owns the records?

# Outline

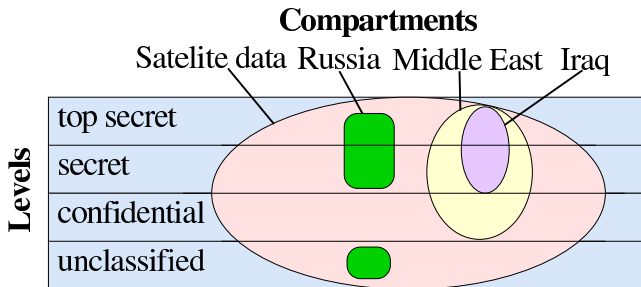
- 1 Motivations
- 2 Access Control Matrix Model
- 3 RBAC
- 4 DAC
- 5 MAC**
- 6 Non-Interference
- 7 Conclusion

## Mandatory Access Control

- AC decisions formalized (and controlled) by comparing security labels indicating sensitivity/criticality of **objects**, with formal authorization, i.e. security clearances, of **subjects**.  
MAC policies often identified with **multilevel security policies**.
- 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.

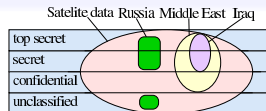


## Labels



- Formalism (+ terminology) comes from US DOD “Orange Book”.
- Formalism combines
  - a **linearly ordered** ranks or sensitivity levels, and
  - a **lattice** of compartments.

## Labels (cont.)



- Historically labels combine classifications on personal and data.

$$Class = \langle rank, compartment \rangle$$

- Dominance relation defined componentwise.

$$(r_1, c_1) \leq (r_2, c_2) :\Leftrightarrow r_1 \leq r_2 \wedge c_1 \subseteq c_2$$

Example:  $(secret, Iraq) \leq (top\ secret, Middle\ East)$

- Authorization** based on comparing labels. For example, a subject with top secret clearance for the Middle East may access (**read**) confidential data on Iraq.
- Well-suited for implementing mandatory **need-to-know** policies, where each subject is assigned a label reflecting **least privilege** required for his function.

## Why lattices?

**Def:** 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.

$$\begin{aligned} a \leq u, b \leq u & \quad \text{and } (a \leq v \ \& \ b \leq v) \rightarrow (u \leq v) \text{ for all } v \in L \\ l \leq a, l \leq b & \quad \text{and } (k \leq a \ \& \ k \leq b) \rightarrow (k \leq l) \text{ for all } k \in L \end{aligned}$$

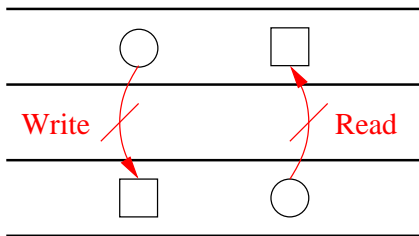
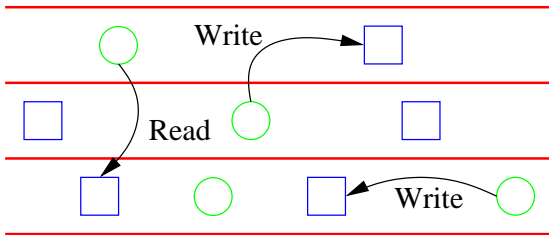
If security labels form a lattice, we can uniquely answer questions like:

- 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**.  
Concerns authorization for subjects **reading** and **writing** objects.
  - Combines state-transition systems with partial-orders on labels.
  - Access decisions satisfy following properties:
    - 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



## BLP (cont.)

- MAC: labels cannot be changed. No information leakage possible!  
No-read-up and no-write-down prevent untrusted subjects from simultaneously having read access to information at one level and write access to information at a lower level.
- But also prevents “legitimate” communication from high-level subjects to low-level ones. Possible solutions:

## BLP (cont.)

- MAC: labels cannot be changed. No information leakage possible!  
No-read-up and no-write-down prevent untrusted subjects from simultaneously having read access to information at one level and write access to information at a lower level.
- 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 \*-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.
  - ⇒ you may only write below your classification and read above it.
- Examples:
  - A manager can overwrite subordinate's data.
  - A monk may write a prayer book read by commoners, but not by a high-priest (whose purer thoughts should not be soiled).
- 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

- 1 Motivations
- 2 Access Control Matrix Model
- 3 RBAC
- 4 DAC
- 5 MAC
- 6 Non-Interference**
- 7 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 & Meseger) : semantic definition of absence of information leakage.

# Non-interference

The basic scenario:

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

## Explicit leak

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

# Non-interference

The basic scenario:

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

## Explicit leak

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

## Implicit leak

Program **if**  $h_1 = h_2$  **then**  $\ell := 0$  **else**  $\ell := 1$  **fi** leaks: the final value of  $\ell$  depends on the value of  $h_1$  and  $h_2$ .

# Non-interference

## 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

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

$A_L = \{a_L \mid a \in A\}$ , i.e.  $\forall S_1 \in f(S) \exists S_2 \in f(S') \cdot S_{1L} = S_{2L}$  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:

$$(l := (h1 = h2)) \oplus_{\frac{1}{2}} (l := 0 \oplus_{\frac{1}{2}} l := 1)$$

## 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  $\ell = 1$

## 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  $P[\ell = 1] = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$

## Cryptographic non-interference

The denotation of a program is a function  $f : \Sigma \rightarrow \mathcal{D}(\Sigma)$  wich 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].
- 4 Semantic security.
- 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 occurences.

## Problem statement

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

### Methods to ensure Non-Interference

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

## Motivating Example (cf. [Volpano'00])

Encryption does not guarantee absence of leakage *per se*:

```

 $l := 0^\eta; m := 0^{\eta-1}1;$ 
for  $i := \eta$  to 1 do    $l_1 := \text{Enc}(k, h|m);$ 
                            $l_2 := \text{Enc}(k, h);$ 
                           if  $(l_1 = l_2)$  then  $l := l|m$  else skip fi ;
                            $m := m \ll 1$  od

```

( $m, l, l_1, l_2$  low-security variables,  $k, h$  high-security variables,  
parameter  $\eta$  size of blocks)

## Example-cntd.

```

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

```

## Example cntd

```

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

```

## 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:

$$\begin{array}{l}
 h_1 := h_2 \\
 h_1 := \ell_1 \\
 h_1 := \ell_1 \oplus h_2 \\
 \ell_1 := h_1 \oplus h_2 \quad ?
 \end{array}
 \quad
 \begin{array}{l}
 \frac{(x, \tau) \in \Gamma \quad \Gamma \vdash e : \tau}{\Gamma \vdash x := e : \tau} \\
 \frac{\Gamma \vdash e : \tau \quad \tau \sqsubseteq \tau'}{\Gamma \vdash e : \tau'} \quad L \sqsubseteq H \\
 \frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 \oplus e_2 : \tau}
 \end{array}$$

# A type system for a simple imperative language

Expressions:

$$\frac{(x, \tau) \in \Gamma}{\Gamma \vdash x : \tau}$$

Sub-typing:  $L \sqsubseteq H$

$$\frac{\Gamma \vdash S : \tau' \quad \tau \sqsubseteq \tau'}{\Gamma \vdash S : \tau}$$

Commands:

$$\frac{(x, \tau) \in \Gamma \quad \Gamma \vdash e : \tau}{\Gamma \vdash x := e}$$

$$\frac{\Gamma \vdash S_1 : \tau \quad \Gamma \vdash S_2 : \tau}{\Gamma \vdash S_1; S_2 : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash f(e_1, e_2) : \tau}$$

$$\frac{\Gamma \vdash e : \tau \quad \tau \sqsubseteq \tau'}{\Gamma \vdash e : \tau'}$$

$$\frac{\Gamma \vdash e : \tau \quad \Gamma \vdash S_1 : \tau \quad \Gamma \vdash S_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } S_1 \text{ else } S_2 \text{ fi} : \tau}$$

$$\frac{\Gamma \vdash e : \tau \quad \Gamma \vdash S : \tau}{\Gamma \vdash \text{while } e \text{ do } S \text{ od} : \tau}$$

## Objectives

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

Computationally Sound Typing for Non-interference: The Case of Deterministic Encryption, by Judicaël Courant and Cristian Ene and Yassine Lakhnech, at FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science, 27th International Conference, New Delhi, India, December 12-14, 2007.

# Outline

- 1 Motivations
- 2 Acces Control Matrix Model
- 3 RBAC
- 4 DAC
- 5 MAC
- 6 Non-Interference
- 7 Conclusion**

# Summary

## Today

- Acces Control
- Matrix Model
- MAC
- DAC
- Non-Interference

## Next Time

- Link between symbolic and computational world.
- CryptoVerif ...
- Funny introduction to Zero Knowledge.

Thank you for your attention



Questions ?