

# Introduction to Information Flow Security

Ana Matos

Pedro Adão

Miguel Correia



# who ARE YOU?

- Ana Almeida Matos
  - teaching this course since 2010
  - research on language based security and analysis of concurrent and distributed programs
- Contact hours: see web page or send email.

# Software Security questions

# Software Security questions

- How can build software that is secure-by-design?

# Software Security questions

- How can build software that is secure-by-design?
- What forms of software vulnerabilities are there and how can they be exploited?

# Software Security questions

- How can build software that is secure-by-design?
- What forms of software vulnerabilities are there and how can they be exploited?

Know the problems

# Software Security questions

- How can build software that is secure-by-design?
- What forms of software vulnerabilities are there and how can they be exploited?
- What are the fundamental mechanics behind the vulnerabilities?

Know the problems

# Software Security questions

- How can build software that is secure-by-design?

- What forms of software vulnerabilities are there and how can they be exploited?



Know the problems

- What are the fundamental mechanics behind the vulnerabilities?




Understand their basic principles



# Software Security questions

- How can build software that is secure-by-design?
- What forms of software vulnerabilities are there and how can they be exploited?  Know the problems
- What are the fundamental mechanics behind the vulnerabilities?  Understand their basic principles
- How can we design techniques and tools to prevent or fix them?

# Software Security questions

- How can build software that is secure-by-design?
- What forms of software vulnerabilities are there and how can they be exploited?  Know the problems
- What are the fundamental mechanics behind the vulnerabilities?  Understand their basic principles
- How can we design techniques and tools to prevent or fix them?  Know how to conceive solutions

# Language-Based Security

# Language-Based Security

- Software applications, system models, security policies, are encoded in programming languages

# Language-Based Security

- Software applications, system models, security policies, are encoded in programming languages
- programming language techniques including semantics, types, optimization and verification, can provide stronger guarantees than programming style guidelines

# Language-Based Security

- Software applications, system models, security policies, are encoded in programming languages
- programming language techniques including semantics, types, optimization and verification, can provide stronger guarantees than programming style guidelines
- security-by-design: using language-based techniques to enforce specified security properties with strong guarantees

# Language Based Information Flow Analysis



- Attacks: Application level
- Tools: Programming language analysis techniques.



- Buffer overflows...

## *Buffer-Overflow Protection: The Theory*

by Krerk Piromsopa and Richard J. Enbody (2006)

Abstract. We propose a framework for protecting against buffer overflow attacks—the oldest and most pervasive attack technique. The malicious nature of buffer-overflow attacks is the use of external data (input) as addresses (or control data). With this observation, we establish a sufficient condition for preventing buffer-overflow attacks and prove that it creates a secure system with respect to buffer-overflow attacks. The underlying concept is that input is untrustworthy, and should not be used as addresses (return addresses and function pointers.). If input can be identified, buffer-overflow attacks can be caught. We used this framework to create an effective, hardware, buffer-overflow prevention tool.



- Pointer integrity...

## *Code-Pointer Integrity*

by Volodymyr Kuznetsov, Laszló Szekerés, Mathias Payer, George Candea, R. Sekar, Dawn Song (2014)

Abstract. Systems code is often written in low-level languages like C/C++, which offer many benefits but also delegate memory management to programmers. This invites memory safety bugs that attackers can exploit to divert control flow and compromise the system. Deployed defense mechanisms (e.g., ASLR, DEP) are incomplete, and stronger defense mechanisms (e.g., CFI) often have high overhead and limited guarantees [19, 15, 9]. We introduce code-pointer integrity (CPI), a new design point that guarantees the integrity of all code pointers in a program (e.g., function pointers, saved return addresses) and thereby prevents all control-flow hijack attacks, including return-oriented programming. (...)

- Format strings...

## *Detecting Format String Vulnerabilities with Type Qualifiers Flow Abstraction*

by Umesh Shankar Kunal Talwar Jeffrey S. Foster David Wagner (2001)  
Abstract. We present a new system for automatically detecting format string security vulnerabilities in C programs using a constraint-based type-inference engine. We describe new techniques for presenting the results of such an analysis to the user in a form that makes bugs easier to find and to fix. The system has been implemented and tested on several real-world software packages. Our tests show that the system is very effective, detecting several bugs previously unknown to the authors and exhibiting a low rate of false positives in almost all cases. Many of our techniques are applicable to additional classes of security vulnerabilities, as well as other type- and constraint based systems.

- Data races...

## *Maximal Sound Predictive Race Detection with Control Flow Abstraction*

by Jeff Huang Patrick O'Neil Meredith Grigore Rosu (2014)

Abstract. Despite the numerous static and dynamic program analysis techniques in the literature, data races remain one of the most common bugs in modern concurrent software. Further, the techniques that do exist either have limited detection capability or are unsound, meaning that they report false positives. We present a sound race detection technique that achieves a provably higher detection capability than existing sound techniques. (...) Moreover, we formally prove that our formulation achieves the maximal possible detection capability (...). We demonstrate via extensive experimentation that our technique detects more races than the other state-of-the-art sound race detection techniques, and that it is scalable to executions of real world concurrent applications with tens of millions of critical events. These experiments also revealed several previously unknown races in real systems (e.g., Eclipse) that have been confirmed or fixed by the developers. (...)

- Code-injection...

*Automated Code Injection Prevention for Web Applications*

by Zhengqin Luo, Tamara Rezk, and Manuel Serrano (2011)

Abstract. We propose a new technique based on multitier compilation for preventing code injection in web applications. It consists in adding an extra stage to the client code generator which compares the dynamically generated code with the specification obtained from the syntax of the source program. No intervention from the programmer is needed. No plugin or modification of the web browser is required. The soundness and validity of the approach are proved formally by showing that the client compiler can be fully abstract. The practical interest of the approach is proved by showing the actual implementation in the Hop environment.

# Design of Enforcement Mechanisms

# Design of Enforcement Mechanisms

- Definition of security properties

# Design of Enforcement Mechanisms

- Definition of security properties
- Static analysis for Security

# Design of Enforcement Mechanisms

- Definition of security properties
- Static analysis for Security
- Dynamic analysis for Security



# Design of Enforcement Mechanisms

- Definition of security properties
- Static analysis for Security
- Dynamic analysis for Security
- Program Verification for Security

# Design of Enforcement Mechanisms

- Definition of security properties
- Static analysis for Security
- Dynamic analysis for Security
- Program Verification for Security

**Transversal  
topic:  
Information  
Flow  
security**

# Design of Enforcement Mechanisms

- Definition of security properties  
Noninterference, ...
- Static analysis for Security
- Dynamic analysis for Security
- Program Verification for Security

**Transversal  
topic:  
Information  
Flow  
security**

# Design of Enforcement Mechanisms

- Definition of security properties  
Noninterference, ...
- Static analysis for Security  
Type systems for Noninterference, ...
- Dynamic analysis for Security
- Program Verification for Security

**Transversal  
topic:  
Information  
Flow  
security**

# Design of Enforcement Mechanisms

- Definition of security properties  
Noninterference, ...
- Static analysis for Security  
Type systems for Noninterference, ...
- Dynamic analysis for Security  
Monitors for Noninterference, ...
- Program Verification for Security

**Transversal  
topic:  
Information  
Flow  
security**

# Design of Enforcement Mechanisms

- Definition of security properties  
Noninterference, ...
- Static analysis for Security  
Type systems for Noninterference, ...
- Dynamic analysis for Security  
Monitors for Noninterference, ...
- Program Verification for Security  
Self-composition for Noninterference, ...

**Transversal  
topic:  
Information  
Flow  
security**





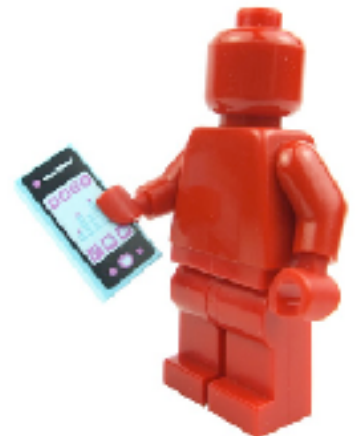




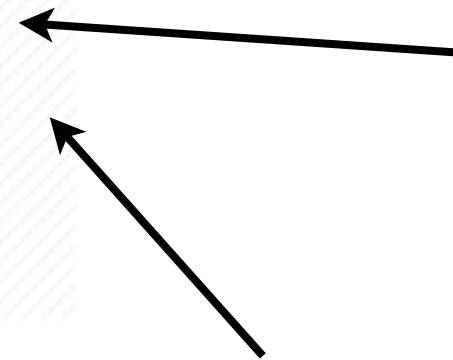
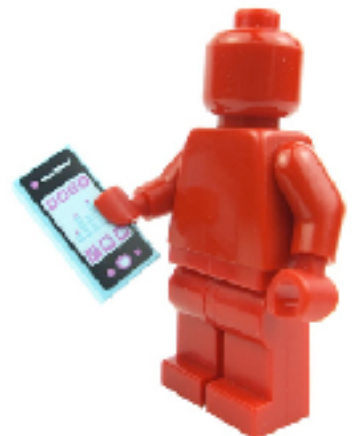
# Which information can go where?



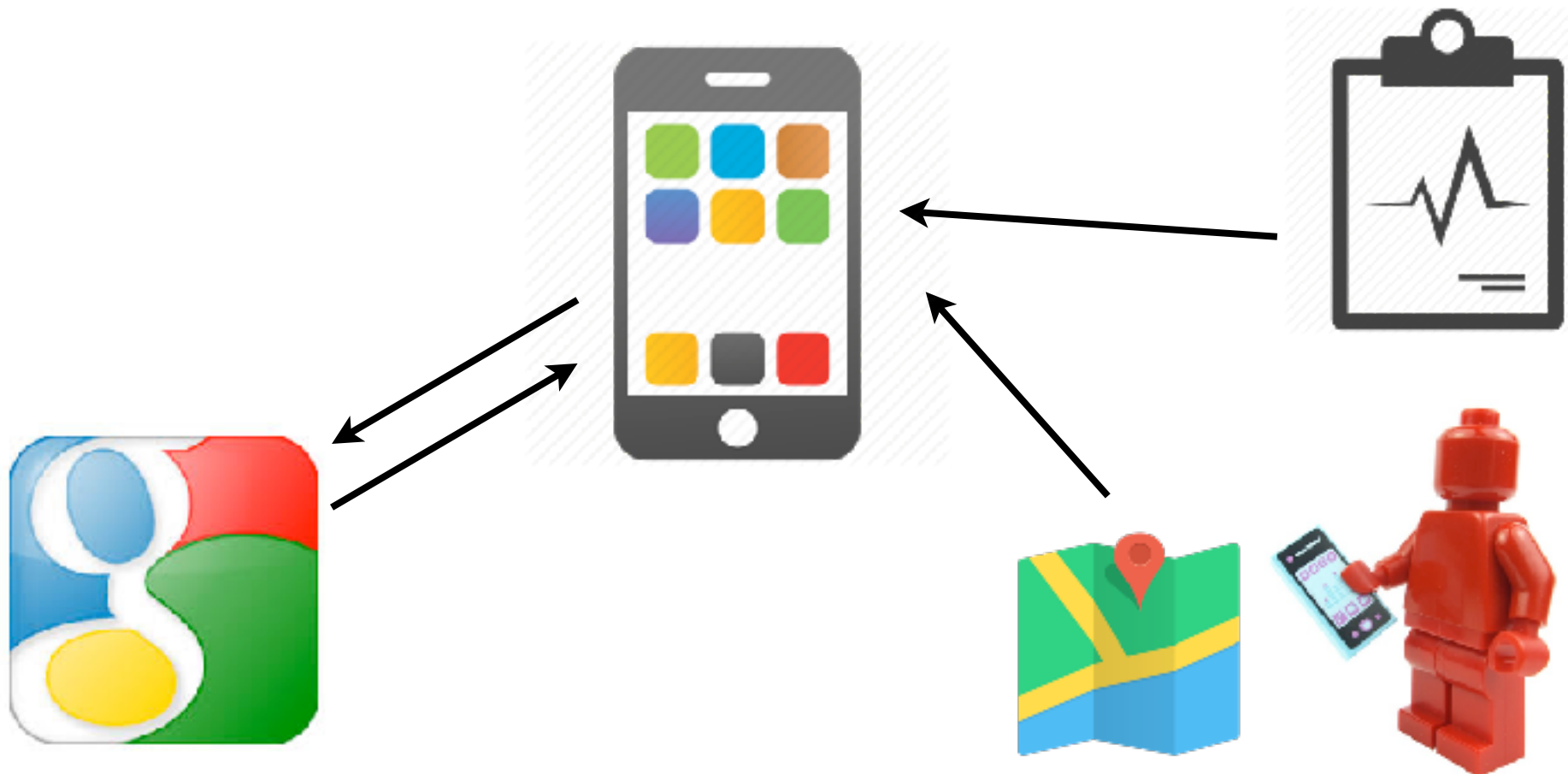
# Which information can go where?



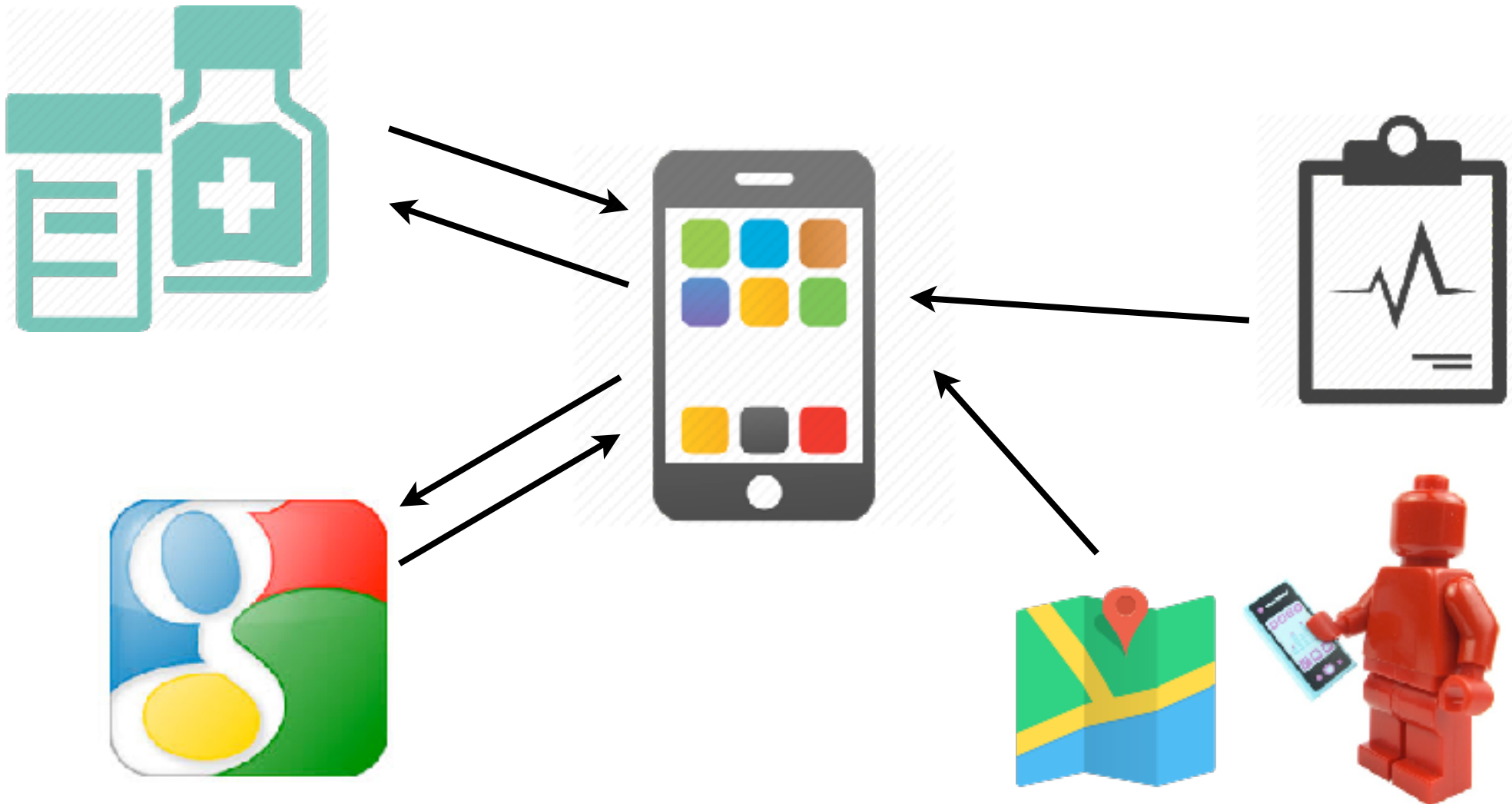
# Which information can go where?



# Which information can go where?



# Which information can go where?



# Class Outline

- Information Flow Security
  - introduction
  - Tracking Information Flow
- Information flow policies

**Is the script “secure”?**



# Is the script “secure”?

Consider a script that

- requests the user’s username and password
- builds and submits an SQL query that retrieves data corresponding to the entered credentials
- outputs the result to the user

Is the script secure?

- It depends on whether the request is trusted.

# Is this script “secure”?

```
1 $a = $_GET[ 'user' ];
2 $b = $_POST[ 'pass' ];
3 $c = "SELECT * FROM users WHERE
      u = '".mysql_real_escape_string($a)."'";
4 $b = "wap";
5 $d = "SELECT * FROM users WHERE u = '". $b. "'";
6 $r = mysql_query($c);
7 $r = mysql_query($d);
8 $b = $_POST[ 'pass' ];
9 $query = "SELECT * FROM users WHERE
          u = '". $a. "' AND p = '". $b. "'";
10 $r = mysql_query($query);
```

The attacker can determine the value of \$a and \$b.

```
1 $a = $_GET[ 'user' ];
2 $b = $_POST[ 'pass' ];
3 $c = "SELECT * FROM users WHERE
      u = '".mysql_real_escape_string($a)."'";
4 $b = "wap";
5 $d = "SELECT * FROM users WHERE u = '". $b. "'";
6 $r = mysql_query($c);
7 $r = mysql_query($d);
8 $b = $_POST[ 'pass' ];
9 $query = "SELECT * FROM users WHERE
          u = '". $a. "' AND p = '". $b. "'";
10 $r = mysql_query($query);
```

The attacker can influence the value of \$c via that of \$a\$ (although it cannot control it completely due to sanitization).

```
1 $a = $_GET['user'];
2 $b = $_POST['pass'];
3 $c = "SELECT * FROM users WHERE
      u = '".mysql_real_escape_string($a)."'";
4 $b = "wap";
5 $d = "SELECT * FROM users WHERE u = '$b.'";
6 $r = mysql_query($c);
7 $r = mysql_query($d);
8 $b = $_POST['pass'];
9 $query = "SELECT * FROM users WHERE
           u = '$a.' AND p = '$b.'";
10 $r = mysql_query($query);
```

The value of \$b is no longer influenced by the attacker.

```
1 $a = $_GET['user'];
2 $b = $_POST['pass'];
3 $c = "SELECT * FROM users WHERE
      u = '".mysql_real_escape_string($a)."'";
4 $b = "wap";
5 $d = "SELECT * FROM users WHERE u = '$b.'";
6 $r = mysql_query($c);
7 $r = mysql_query($d);
8 $b = $_POST['pass'];
9 $query = "SELECT * FROM users WHERE
           u = '$a.' AND p = '$b.'";
10 $r = mysql_query($query);
```

The value of \$r is influenced by the attacker via that of \$c.

```
1 $a = $_GET['user'];
2 $b = $_POST['pass'];
3 $c = "SELECT * FROM users WHERE
      u = '".mysql_real_escape_string($a)."'";
4 $b = "wap";
5 $d = "SELECT * FROM users WHERE u = '". $b. "'";
6 $r = mysql_query($c);
7 $r = mysql_query($d);
8 $b = $_POST['pass'];
9 $query = "SELECT * FROM users WHERE
          u = '". $a. "' AND p = '". $b. "'";
10 $r = mysql_query($query);
```

The value of \$r is no longer influenced by the attacker.

```
1 $a = $_GET['user'];
2 $b = $_POST['pass'];
3 $c = "SELECT * FROM users WHERE
      u = '".mysql_real_escape_string($a)."'";
4 $b = "wap";
5 $d = "SELECT * FROM users WHERE u = '". $b. "'";
6 $r = mysql_query($c);
7 $r = mysql_query($d);
8 $b = $_POST['pass'];
9 $query = "SELECT * FROM users WHERE
          u = '". $a. "' AND p = '". $b. "'";
10 $r = mysql_query($query);
```

The attacker can determine the value of \$b.

```
1 $a = $_GET['user'];
2 $b = $_POST['pass'];
3 $c = "SELECT * FROM users WHERE
      u = '".mysql_real_escape_string($a)."'";
4 $b = "rap";
5 $d = "SELECT * FROM users WHERE u = '$b.'";
6 $r = mysql_query($c);
7 $r = mysql_query($d);
8 $b = $_POST['pass'];
9 $query = "SELECT * FROM users WHERE
           u = '$a.' AND p = '$b.'";
10 $r = mysql_query($query);
```



The attacker can determine the value of \$query via those of \$a and \$b.

```
1 $a = $_GET['user'];
2 $b = $_POST['pass'];
3 $c = "SELECT * FROM users WHERE
      u = '".mysql_real_escape_string($a)."'";
4 $b = "wa";
5 $d = "SELECT * FROM users WHERE u = '$b.'";
6 $r = mysql_query($c);
7 $r = mysql_query($d);
8 $b = $_POST['pass'];
9 $query = "SELECT * FROM users WHERE
          u = '$a.' AND p = '$b.'";
10 $r = mysql_query($query);
```

The attacker can tamper the query that is placed.

```
1 $a = $_GET['user'];
2 $b = $_POST['pass'];
3 $c = "SELECT * FROM users WHERE
      u = '".mysql_real_escape_string($a)."'";
4 $b = "wap";
5 $d = "SELECT * FROM users WHERE u = '$b.'";
6 $r = mysql_query($c);
7 $r = mysql_query($d);
8 $b = $_POST['pass'];
9 $query = "SELECT * FROM users WHERE
          u = '$a.' AND p = '$b.'";
10 $r = mysql_query($query);
```

# Perl's Taint Mode

# Perl's Taint Mode

- While in **Taint mode** (either when program running with differing real and effective IDs, or by using '-T'), Perl performs taint checks:

# Perl's Taint Mode

- While in **Taint mode** (either when program running with differing real and effective IDs, or by using '-T'), Perl performs taint checks:
- All forms of input to the programs are marked as "**tainted**".

# Perl's Taint Mode

- While in **Taint mode** (either when program running with differing real and effective IDs, or by using '-T'), Perl performs taint checks:
- All forms of input to the programs are marked as “**tainted**”.
- Tainted variables **taint** variables explicitly calculated from them

# Perl's Taint Mode

- While in **Taint mode** (either when program running with differing real and effective IDs, or by using '-T'), Perl performs taint checks:
- All forms of input to the programs are marked as **"tainted"**.
- Tainted variables **taint** variables explicitly calculated from them
- Tainted data may not be used in any **sensitive command** (with some exceptions).

# Implicit in this mechanism

- A **set of security classes** (ex:Tainted vs. Untainted)
- A **classification** of objects/information holders (ex: variables, input channels, sensitive sinks)
- A specification of when information **can flow** from one security class to another (ex:Tainted  $\rightarrow$  Untainted)
- A way to determine security classes that safely represent the **combination** of two other (ex: taintedness is “absorbing”)



# Class Outline

# Class Outline

- Information Flow Security
  - introduction

# Class Outline

- Information Flow Security
  - introduction
- Tracking Information Flow

# Class Outline

- Information Flow Security
  - introduction
- Tracking Information Flow
- Information flow policies

# Information flow policies



“Lattice-Based Access Control Models”, R. Sandhu, 1993.

# Security policies

- Information Security “goals”:
  - **Confidentiality**  
prevent/detect *undesired*  
disclosure of information
  - **Integrity**  
prevent/detect *undesired*  
modification of information
  - **Availability**  
ensure readiness for *desirable*  
accesses

# Security policies

- Information Security “goals”:
  - **Confidentiality**  
prevent/detect *undesired* disclosure of information
  - **Integrity**  
prevent/detect *undesired* modification of information
  - **Availability**  
ensure readiness for *desirable* accesses

# Security policies

- Information Security “goals”:
  - **Confidentiality**  
prevent/detect *undesired* ?  
disclosure of information
  - **Integrity**  
prevent/detect *undesired* ?  
modification of information
  - **Availability**  
ensure readiness for *desirable*  
accesses



# Security policies

- Information Security “goals”:
  - **Confidentiality**  
prevent/detect *undesired* ?  
disclosure of information
  - **Integrity**  
prevent/detect *undesired* ?  
modification of information
  - **Availability**  
ensure readiness for *desirable* ?  
accesses

# Security policies

- Information Security “goals”:

- **Confidentiality**

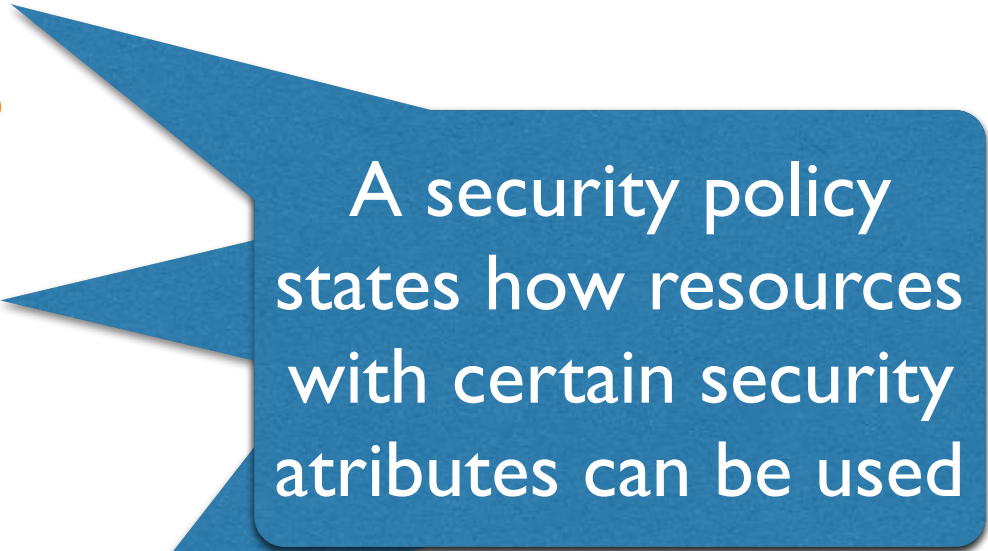
prevent/detect *undesired*  
disclosure of information

- **Integrity**

prevent/detect *undesired*  
modification of information

- **Availability**

ensure readiness for *desirable*  
accesses



A security policy  
states how resources  
with certain security  
attributes can be used

To define an  
Information flow policy

# To define an Information flow policy

- We choose

# To define an Information flow policy

- We choose
  - a set of security classes

# To define an Information flow policy

- We choose
  - a set of security classes
  - a can-flow relation between them,

# To define an Information flow policy

- We choose
  - a set of security classes
  - a can-flow relation between them,
  - and an operator for combining them.

# Information Flow Policies for Confidentiality



# Information Flow Policies for Confidentiality

- Security classes determine who has the right to **read**.

# Information Flow Policies for Confidentiality

- Security classes determine who has the right to **read**.
- Information can only flow towards confidentiality classes that are **at least as secret**.

# Information Flow Policies for Confidentiality

- Security classes determine who has the right to **read**.
- Information can only flow towards confidentiality classes that are **at least as secret**.
- Information that is derived from the combination of two security classes takes a confidentiality classes that are **at least as secret** as each of them.

# Information Flow Policies for Integrity

# Information Flow Policies for Integrity

- Security classes determine who has the right to **write**.

# Information Flow Policies for Integrity

- Security classes determine who has the right to **write**.
- Information can only flow towards integrity classes that are **no more trustful**.

# Information Flow Policies for Integrity

- Security classes determine who has the right to **write**.
- Information can only flow towards integrity classes that are **no more trustful**.
- Information that is derived from the combination of two integrity classes takes an integrity class that is **no more trustful** than each of them.

# Confidentiality classes



# Confidentiality classes

**SECRET**

# Confidentiality classes

**SECRET**

**TOP  
SECRET**

# Confidentiality classes

**SUPER  
TOP SECRET**

**SECRET**

**TOP  
SECRET**

# Confidentiality classes

**SUPER  
TOP SECRET**

**SECRET**

**ULTRA TOP  
SECRET**

**TOP  
SECRET**

# Confidentiality classes

**SUPER  
TOP SECRET**

**SECRET**

**ULTRA TOP  
SECRET**

**TOP  
SECRET**

**UNCLASSIFIED**

# Confidentiality classes

**SUPER  
TOP SECRET**

**SECRET**

**TOP  
SECRET**

**ULTRA TOP  
SECRET**

**CLASSIFIED**

**UNCLASSIFIED**

# Confidentiality classes

**SUPER  
TOP SECRET**

**SECRET**

**TOP  
SECRET**

**ULTRA TOP  
SECRET**

**CLASSIFIED**

**CONFIDENTIAL**

**UNCLASSIFIED**



# Confidentiality classes

**SUPER  
TOP SECRET**

**SECRET**

**TOP  
SECRET**

**ULTRA TOP  
SECRET**

**CLASSIFIED**

**CONFIDENTIAL**

**PUBLIC**

**UNCLASSIFIED**



# Confidentiality classes

**SUPER  
TOP SECRET**

**SECRET**

**TOP  
SECRET**

**ULTRA TOP  
SECRET**

**CLASSIFIED**

**CONFIDENTIAL**

**FULL  
DISCLOSURE**

**PUBLIC**

**UNCLASSIFIED**

# Confidentiality classes



# Confidentiality classes



# Integrity classes



# Integrity classes



# Integrity classes



# Integrity classes



# Integrity classes

**TRUSTED**



**RELIABLE**

**CORRUPT**



# Integrity classes



# Integrity classes



# Integrity classes





# Integrity classes



# Integrity classes



# Example: principal-based policy

# Example: principal-based policy

- Principal-based **security classes** - classes are sets of principals:

# Example: principal-based policy

- Principal-based **security classes** - classes are **sets of principals**:
- Confidentiality - class {Alice, Bob} means that Alice and Bob are allowed to **read**.



# Example: principal-based policy

- Principal-based **security classes** - classes are **sets of principals**:
- Confidentiality - class {Alice, Bob} means that Alice and Bob are allowed to **read**.
- Integrity - class {Alice, Bob} means that Alice and Bob are allowed to **write**.

# Can-flow relation

**SECRET**

**TRUSTED**

**PUBLIC**

**UNRELIABLE**

# Can-flow relation

**SECRET**



**TRUSTED**

**PUBLIC**

**UNRELIABLE**

# Can-flow relation

**SECRET**



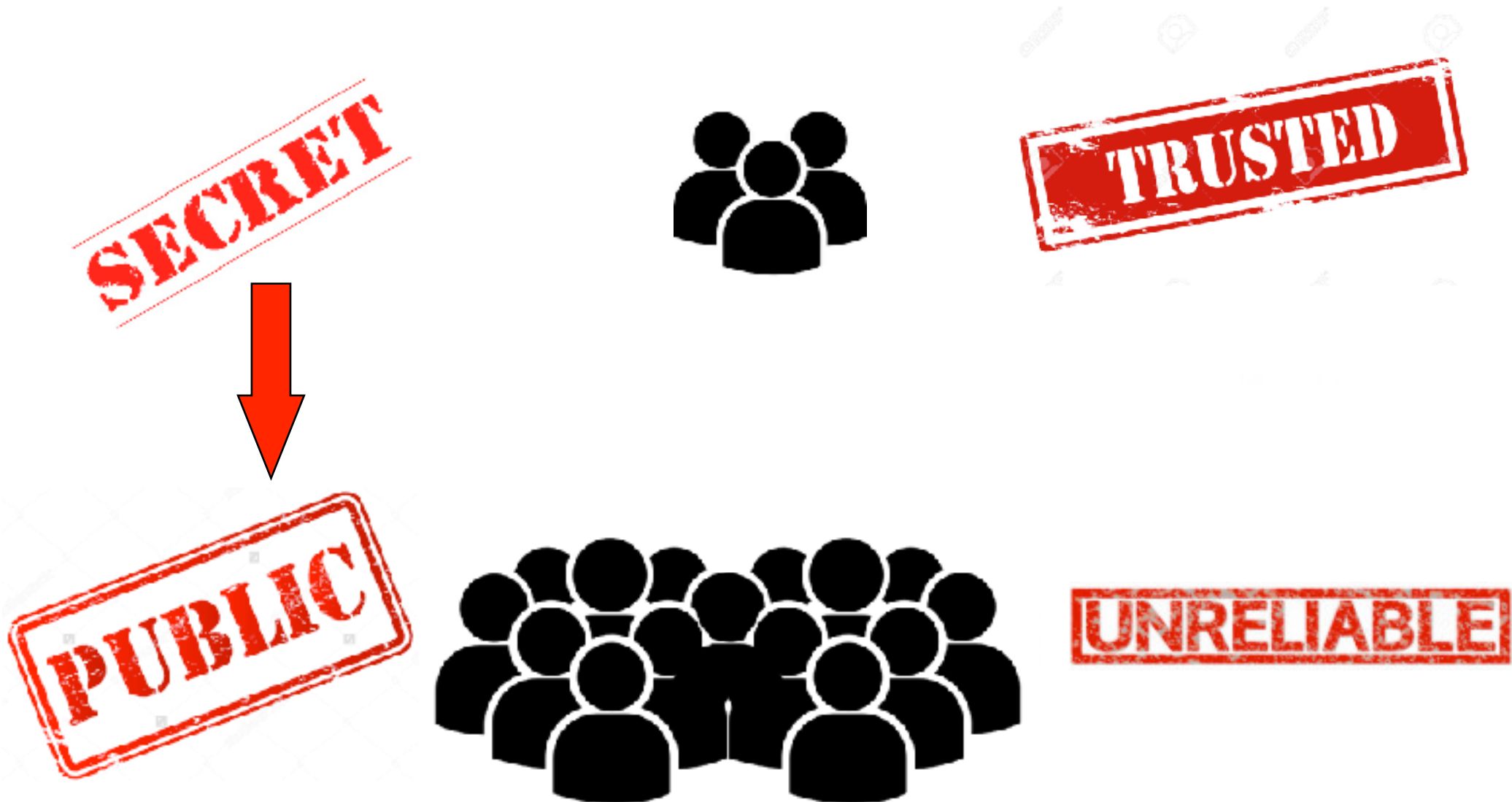
**TRUSTED**

**PUBLIC**

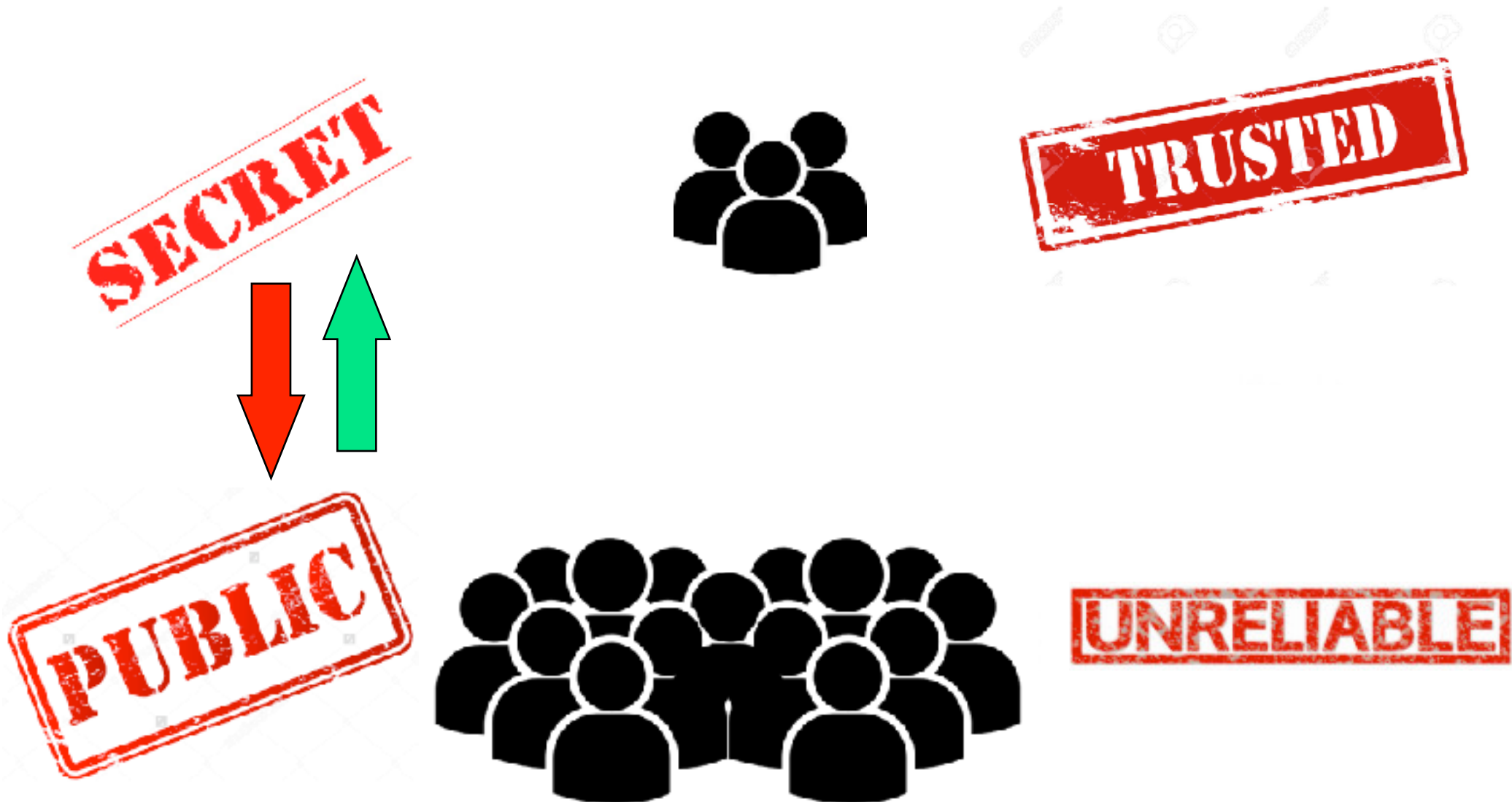


**UNRELIABLE**

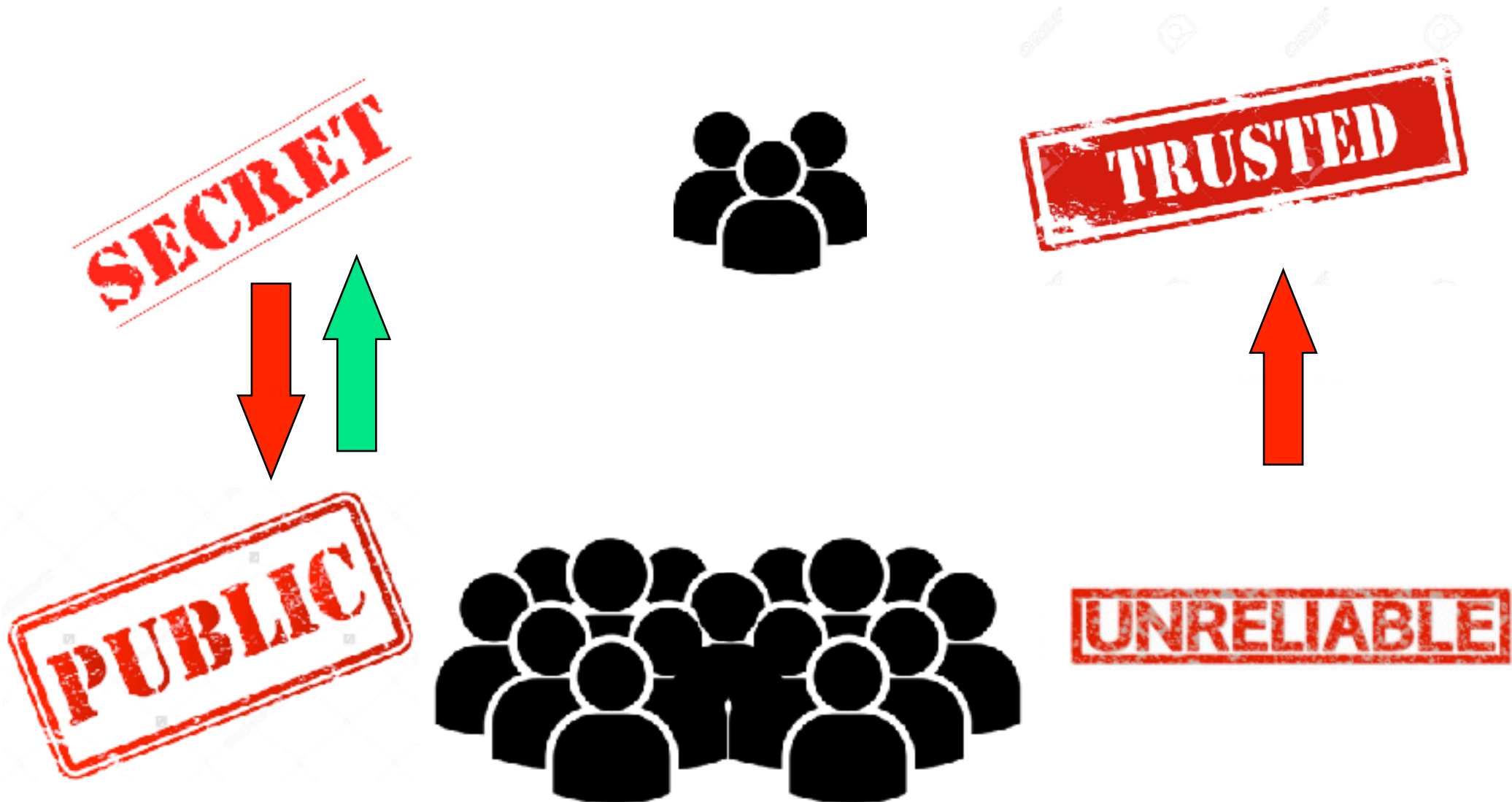
# Can-flow relation



# Can-flow relation

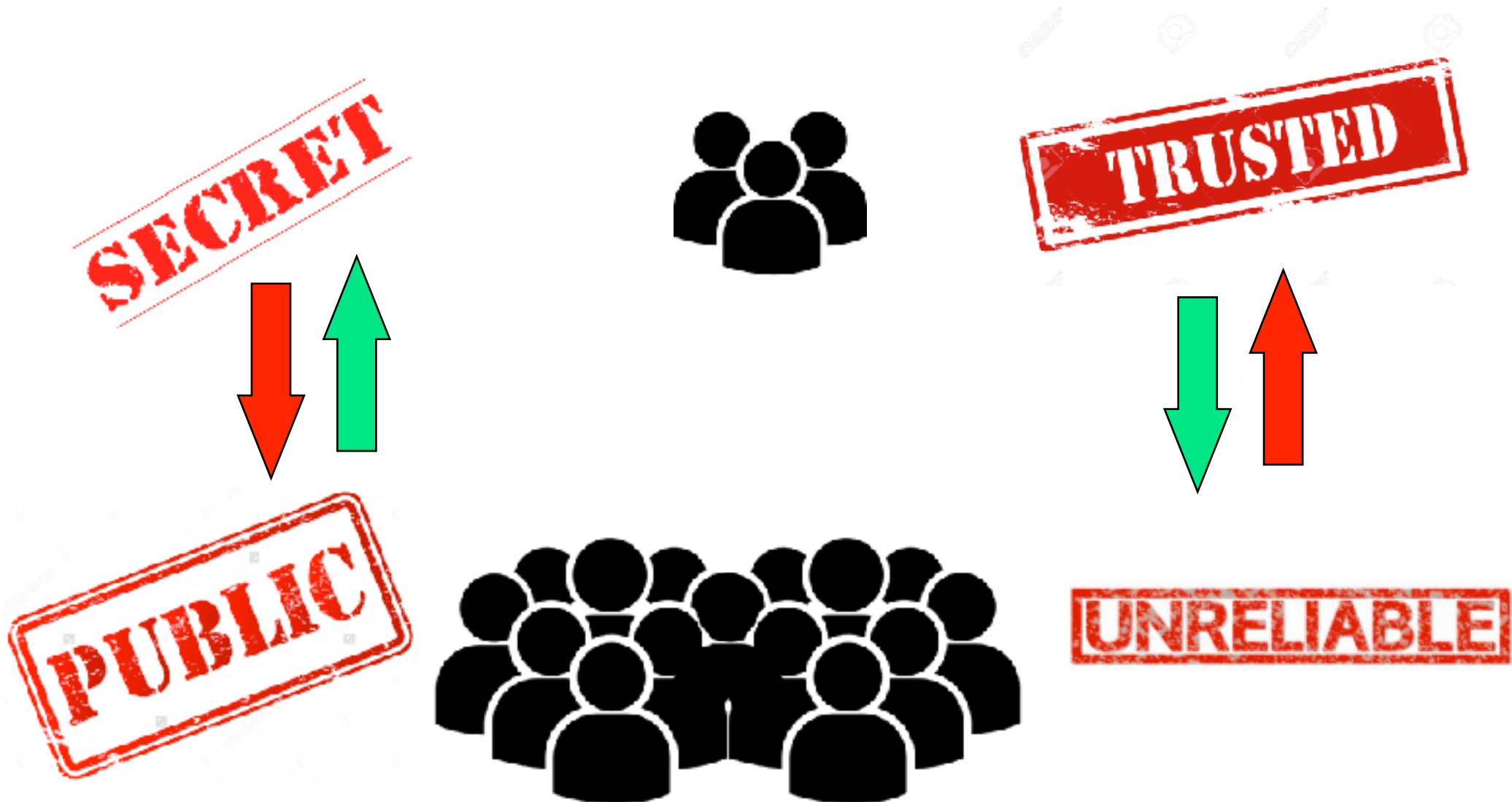


# Can-flow relation





# Can-flow relation





# Example: principal-based policy

# Example: principal-based policy

- Principal-based **can-flow relation**

# Example: principal-based policy

- Principal-based **can-flow relation**
  - Confidentiality - information can flow from a class to another if all the principals in the target class are in the origin class (**subset**).

# Example: principal-based policy

- Principal-based **can-flow relation**
  - Confidentiality - information can flow from a class to another if all the principals in the target class are in the origin class (**subset**).
  - Integrity - information can flow from a class to another if all the principals in the origin class are in the target class (**superset**).

# Class combination (confidentiality)



?

# Class combination (confidentiality)



**SECRET**

?



# Class combination (confidentiality)



**SECRET**

**PUBLIC**

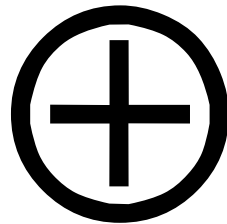
?



# Class combination (confidentiality)



**SECRET**



**PUBLIC**

?

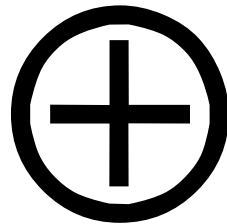




# Class combination (confidentiality)



**SECRET**



**PUBLIC**

**=**

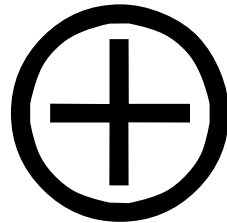
**?**



# Class combination (confidentiality)



**SECRET**



**PUBLIC**



**SECRET**



# Class combination (integrity)



?

# Class combination (integrity)



**UNRELIABLE**

?



# Class combination (integrity)



**UNRELIABLE**

**TRUSTED**

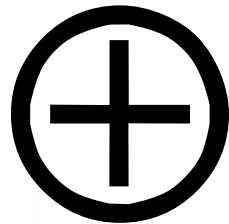
?



# Class combination (integrity)



UNRELIABLE



TRUSTED

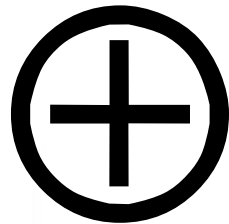
?



# Class combination (integrity)



**UNRELIABLE**



**TRUSTED**

**=**

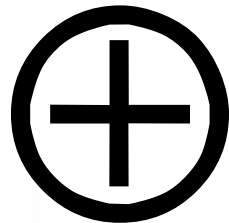
**?**



# Class combination (integrity)



**UNRELIABLE**



**TRUSTED**



**UNRELIABLE**





# Example: principal-based policy

# Example: principal-based policy

- Principal-based **class combination**:

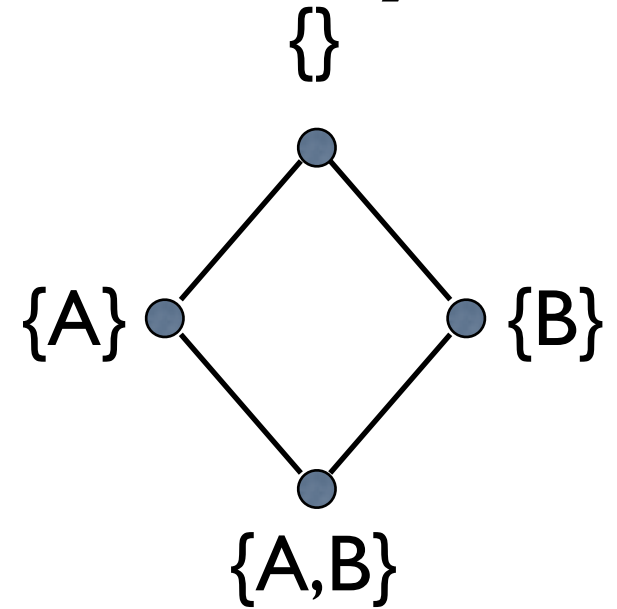
# Example: principal-based policy

- Principal-based **class combination**:
  - Confidentiality - resulting class contains principals that are in both of the origin classes (**intersection**).

# Example: principal-based policy

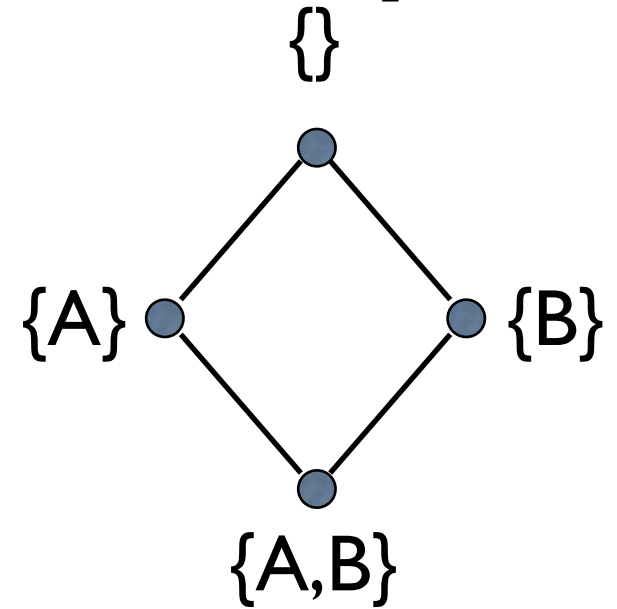
- Principal-based **class combination**:
  - Confidentiality - resulting class contains principals that are in both of the origin classes (**intersection**).
  - Integrity - resulting class contains principals that are in either of the origin classes (**union**).

# Example: Principal-based policy for confidentiality



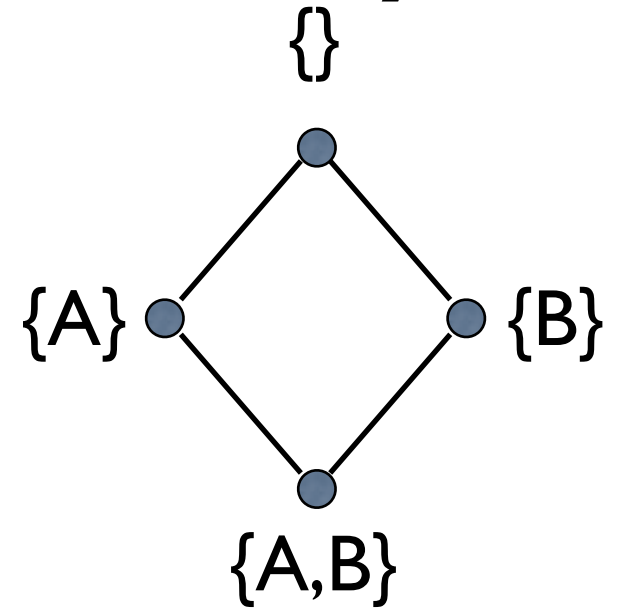
# Example: Principal-based policy for confidentiality

- $SC = \{ \{\}, \{A\}, \{B\}, \{A,B\} \}$



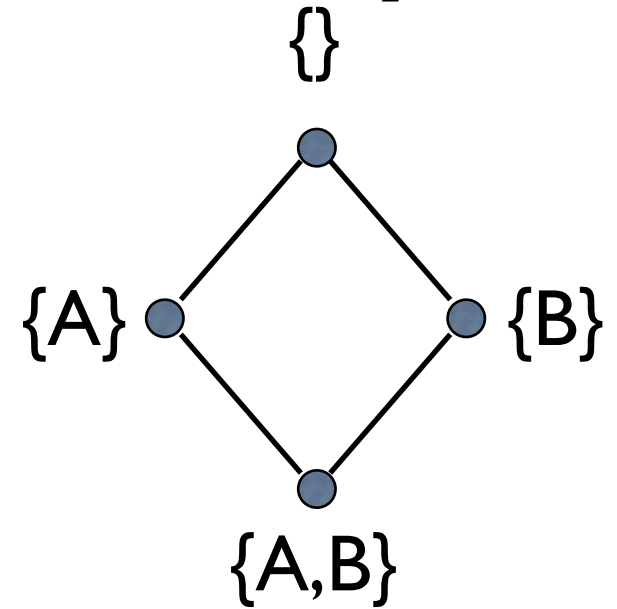
# Example: Principal-based policy for confidentiality

- $SC = \{ \{\}, \{A\}, \{B\}, \{A,B\} \}$
- $\rightarrow = \{(\{A\}, \{\}), (\{B\}, \{\}), (\{A,B\}, \{\}),$   
 $(\{A,B\}, \{A\}), (\{A,B\}, \{B\}),$   
 $(\{\}, \{\}), (\{A\}, \{A\}), (\{B\}, \{B\}), (\{A,B\}, \{A,B\})$   
(or, equivalently,  $\rightarrow = \supseteq$ )



# Example: Principal-based policy for confidentiality

- $SC = \{ \{\}, \{A\}, \{B\}, \{A,B\} \}$
- $\rightarrow = \supseteq$
- 



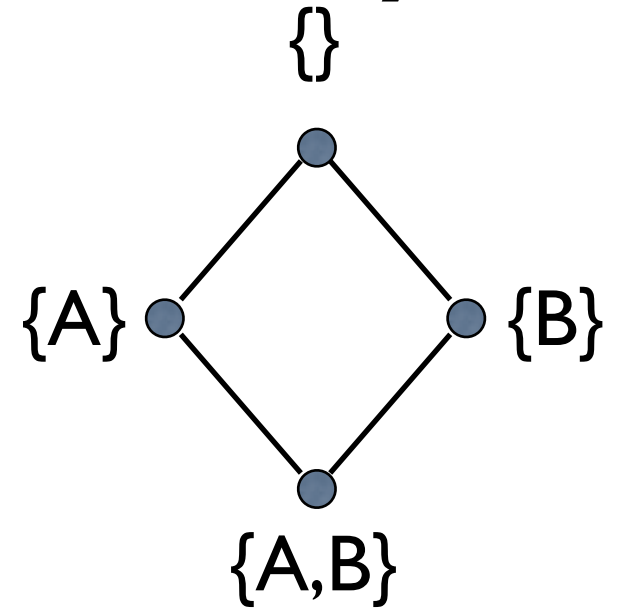


# Example: Principal-based policy for confidentiality

- $SC = \{ \{\}, \{A\}, \{B\}, \{A,B\} \}$

- $\rightarrow = \supseteq$

- $\{\} \oplus \{\} = \{\}, \{A\} \oplus \{A\} = \{A\}, \{B\} \oplus \{B\} = \{B\}, \{A,B\} \oplus \{A,B\} = \{A,B\},$

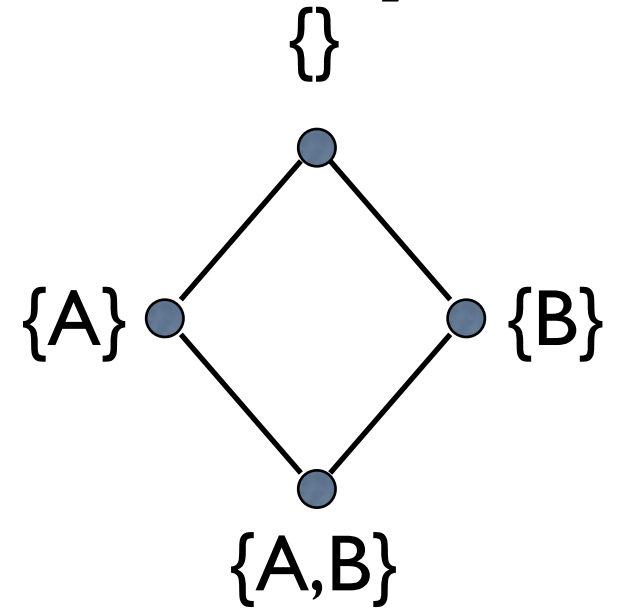


# Example: Principal-based policy for confidentiality

- $SC = \{ \{\}, \{A\}, \{B\}, \{A,B\} \}$

- $\rightarrow = \supseteq$

- $\{\} \oplus \{\} = \{\}, \{A\} \oplus \{A\} = \{A\}, \{B\} \oplus \{B\} = \{B\}, \{A,B\} \oplus \{A,B\} = \{A,B\},$   
 $\{\} \oplus \{A\} = \{\}, \{\} \oplus \{B\} = \{\}, \{\} \oplus \{A,B\} = \{\},$

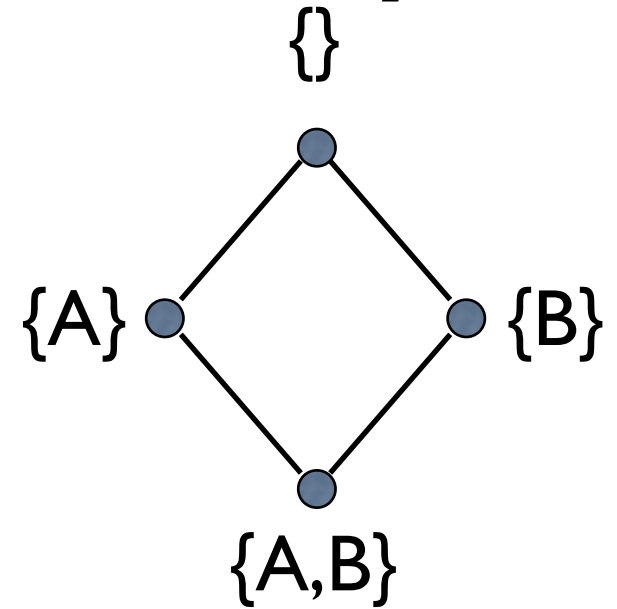


# Example: Principal-based policy for confidentiality

- $SC = \{ \{\}, \{A\}, \{B\}, \{A,B\} \}$

- $\rightarrow = \supseteq$

- $\{\} \oplus \{\} = \{\}, \{A\} \oplus \{A\} = \{A\}, \{B\} \oplus \{B\} = \{B\}, \{A,B\} \oplus \{A,B\} = \{A,B\},$   
 $\{\} \oplus \{A\} = \{\}, \{\} \oplus \{B\} = \{\}, \{\} \oplus \{A,B\} = \{\},$   
 $\{A\} \oplus \{A,B\} = \{A\}, \{B\} \oplus \{A,B\} = \{B\}, \{A\} \oplus \{B\} = \{\}.$

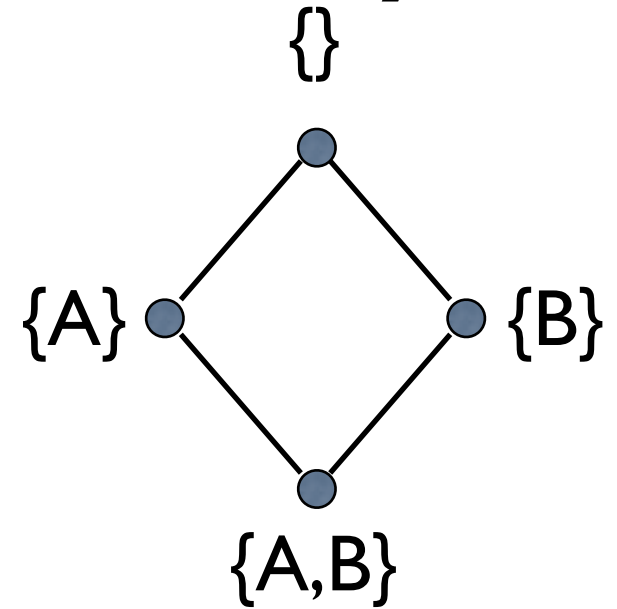


# Example: Principal-based policy for confidentiality

- $SC = \{ \{\}, \{A\}, \{B\}, \{A,B\} \}$

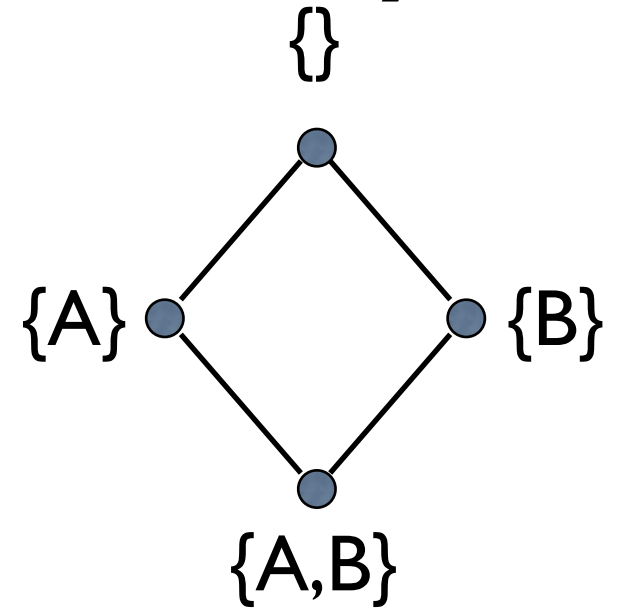
- $\rightarrow = \supseteq$

- $\{\} \oplus \{\} = \{\}, \{A\} \oplus \{A\} = \{A\}, \{B\} \oplus \{B\} = \{B\}, \{A,B\} \oplus \{A,B\} = \{A,B\},$   
 $\{\} \oplus \{A\} = \{\}, \{\} \oplus \{B\} = \{\}, \{\} \oplus \{A,B\} = \{\},$   
 $\{A\} \oplus \{A,B\} = \{A\}, \{B\} \oplus \{A,B\} = \{B\}, \{A\} \oplus \{B\} = \{\}.$   
(or, equivalently,  $\oplus = \cap$ )



# Example: Principal-based policy for confidentiality

- $SC = \{ \{\}, \{A\}, \{B\}, \{A,B\} \}$
- $\rightarrow = \supseteq$
- $\oplus = \cap$



# Information flow policy (formally)

# Information flow policy (formally)

- A triple  $(SC, \rightarrow, \oplus)$  where

# Information flow policy (formally)

- A triple  $(SC, \rightarrow, \oplus)$  where
  - $SC$  is a set of security classes,




# Information flow policy (formally)

- A triple  $(SC, \rightarrow, \oplus)$  where
  - $SC$  is a set of security classes,
  - $\rightarrow \subseteq SC \times SC$  is a binary can-flow relation on  $SC$ ,

# Information flow policy (formally)

- A triple  $(SC, \rightarrow, \oplus)$  where
  - $SC$  is a set of security classes,
  - $\rightarrow \subseteq SC \times SC$  is a binary can-flow relation on  $SC$ ,
  - and  $\oplus : SC \times SC \rightarrow SC$  is an associative and commutative binary class-combining or join operator on  $SC$ .

# Information flow policy (formal)



Which labels can be given to objects.

- A triple  $(SC, \rightarrow, \oplus)$  where
  - $SC$  is a set of security classes,
  - $\rightarrow \subseteq SC \times SC$  is a binary can-flow relation on  $SC$ ,
  - and  $\oplus : SC \times SC \rightarrow SC$  is an associative and commutative binary class-combining or join operator on  $SC$ .

# Information flow policy (formal)

Which labels can be  
given to objects.

- A triple  $(SC, \rightarrow, \oplus)$  where
  - $SC$  is a set of security classes
  - $\rightarrow \subseteq SC \times SC$  is a binary relation on  $SC$ ,
  - and  $\oplus : SC \times SC \rightarrow SC$  is an associative and commutative binary class-combining or join operator on  $SC$ .

How information is  
allowed to flow.

# Information flow policy (formal)

- A triple  $(SC, \rightarrow, \oplus)$  where

- $SC$  is a set of security classes

- $\rightarrow \subseteq SC \times SC$  is a binary relation on  $SC$ ,

- and  $\oplus : SC \times SC \rightarrow SC$  is associative

and commutative binary operator

or join operator on  $SC$

Which labels can be given to objects.

How information is allowed to flow.

What is the security class for describing information coming from two classes

# Example: High-Low policies

# Example: High-Low policies

- High-Low policy for confidentiality:

# Example: High-Low policies

- High-Low policy for confidentiality:
  - $SC = \{H, L\}$



# Example: High-Low policies

- High-Low policy for confidentiality:
  - $SC = \{H, L\}$
  - $\rightarrow = \{(H, H), (L, L), (L, H)\}$

# Example: High-Low policies

- High-Low policy for confidentiality:
  - $SC = \{H, L\}$
  - $\rightarrow = \{(H, H), (L, L), (L, H)\}$
  - $H \oplus H = H, L \oplus H = H, L \oplus L = L$

# Example: High-Low policies

- High-Low policy for confidentiality:
  - $SC = \{H, L\}$
  - $\rightarrow = \{(H, H), (L, L), (L, H)\}$
  - $H \oplus H = H, L \oplus H = H, L \oplus L = L$

# Example: High-Low policies

- High-Low policy for **confidentiality**:
  - $SC = \{H, L\}$
  - $\rightarrow = \{(H, H), (L, L), (L, H)\}$
  - $H \oplus H = H, L \oplus H = H, L \oplus L = L$
- High-Low policy for **integrity**:

# Example: High-Low policies

- High-Low policy for **confidentiality**:
  - $SC = \{H, L\}$
  - $\rightarrow = \{(H, H), (L, L), (L, H)\}$
  - $H \oplus H = H, L \oplus H = H, L \oplus L = L$
- High-Low policy for **integrity**:
  - $SC = \{H, L\}$

# Example: High-Low policies

- High-Low policy for **confidentiality**:
  - $SC = \{H, L\}$
  - $\rightarrow = \{(H, H), (L, L), (L, H)\}$
  - $H \oplus H = H, L \oplus H = H, L \oplus L = L$
- High-Low policy for **integrity**:
  - $SC = \{H, L\}$
  - $\rightarrow = \{(H, H), (L, L), (H, L)\}$

# Example: High-Low policies

- High-Low policy for **confidentiality**:
  - $SC = \{H, L\}$
  - $\rightarrow = \{(H, H), (L, L), (L, H)\}$
  - $H \oplus H = H, L \oplus H = H, L \oplus L = L$
- High-Low policy for **integrity**:
  - $SC = \{H, L\}$
  - $\rightarrow = \{(H, H), (L, L), (H, L)\}$
  - $H \oplus H = H, L \oplus H = L, L \oplus L = L$

Example from previous classes:  
root-user policies



# Example from previous classes: root-user policies

- Root-user policy for confidentiality:

# Example from previous classes: root-user policies

- Root-user policy for confidentiality:
  - Confidentiality Classes: root, user

# Example from previous classes: root-user policies

- Root-user policy for confidentiality:
  - Confidentiality Classes: root, user
  - $\text{root} \rightarrow \text{root}, \text{user} \rightarrow \text{user}, \text{user} \rightarrow \text{root}$

# Example from previous classes: root-user policies

- Root-user policy for confidentiality:
  - Confidentiality Classes: root, user
  - $\text{root} \rightarrow \text{root}, \text{user} \rightarrow \text{user}, \text{user} \rightarrow \text{root}$
  - $\text{root} \oplus \text{root} = \text{root}, \text{user} \oplus \text{user} = \text{user},$   
 $\text{user} \oplus \text{root} = \text{root},$

# Example from previous classes: root-user policies

- Root-user policy for confidentiality:
  - Confidentiality Classes: root, user
  - $\text{root} \rightarrow \text{root}, \text{user} \rightarrow \text{user}, \text{user} \rightarrow \text{root}$
  - $\text{root} \oplus \text{root} = \text{root}, \text{user} \oplus \text{user} = \text{user},$   
 $\text{user} \oplus \text{root} = \text{root},$
- Root-user policy for integrity:

# Example from previous classes: root-user policies

- Root-user policy for confidentiality:
  - Confidentiality Classes: root, user
  - $\text{root} \rightarrow \text{root}, \text{user} \rightarrow \text{user}, \text{user} \rightarrow \text{root}$
  - $\text{root} \oplus \text{root} = \text{root}, \text{user} \oplus \text{user} = \text{user},$   
 $\text{user} \oplus \text{root} = \text{root},$
- Root-user policy for integrity:
  - Integrity Classes: root, user

# Example from previous classes: root-user policies

- Root-user policy for confidentiality:
  - Confidentiality Classes: root, user
  - $\text{root} \rightarrow \text{root}, \text{user} \rightarrow \text{user}, \text{user} \rightarrow \text{root}$
  - $\text{root} \oplus \text{root} = \text{root}, \text{user} \oplus \text{user} = \text{user},$   
 $\text{user} \oplus \text{root} = \text{root},$
- Root-user policy for integrity:
  - Integrity Classes: root, user
  - $\text{root} \rightarrow \text{root}, \text{user} \rightarrow \text{user}, \text{root} \rightarrow \text{user}$

# Example from previous classes: root-user policies

- Root-user policy for confidentiality:
  - Confidentiality Classes: root, user
  - $\text{root} \rightarrow \text{root}, \text{user} \rightarrow \text{user}, \text{user} \rightarrow \text{root}$
  - $\text{root} \oplus \text{root} = \text{root}, \text{user} \oplus \text{user} = \text{user},$   
 $\text{user} \oplus \text{root} = \text{root},$
- Root-user policy for integrity:
  - Integrity Classes: root, user
  - $\text{root} \rightarrow \text{root}, \text{user} \rightarrow \text{user}, \text{root} \rightarrow \text{user}$
  - $\text{root} \oplus \text{root} = \text{root}, \text{user} \oplus \text{root} = \text{user},$   
 $\text{user} \oplus \text{user} = \text{user}$



# Partially ordered policies

# Partially ordered policies

- It often makes sense to assume that:

# Partially ordered policies

- It often makes sense to assume that:
  - Information can always flow within the same security level.

# Partially ordered policies

- It often makes sense to assume that:
  - Information can always flow within the same security level.
  - Two security level that are related to others in the same way are the same security level.

# Partially ordered policies

- It often makes sense to assume that:
  - Information can always flow within the same security level.
  - Two security level that are related to others in the same way are the same security level.
  - If information can flow from A to B and from B to C, it can flow from A to C.

# Partial orders

# Partial orders

- The flow relation  $\rightarrow \subseteq SC \times SC$  is a **partial order**  $(SC, \rightarrow)$  if it is:

# Partial orders

- The flow relation  $\rightarrow \subseteq SC \times SC$  is a **partial order**  $(SC, \rightarrow)$  if it is:
  - **Reflexive:**  
for all  $s \in SC$ ,  $s \rightarrow s$ ; and



# Partial orders

- The flow relation  $\rightarrow \subseteq SC \times SC$  is a **partial order**  $(SC, \rightarrow)$  if it is:
  - **Reflexive:**  
for all  $s \in SC$ ,  $s \rightarrow s$ ; and
  - **Anti-symmetric:**  
 $s_1 \rightarrow s_2$  and  $s_2 \rightarrow s_1$  implies  $s_1 = s_2$ ; and

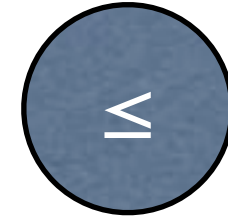
# Partial orders

- The flow relation  $\rightarrow \subseteq SC \times SC$  is a **partial order**  $(SC, \rightarrow)$  if it is:
  - **Reflexive:**  
for all  $s \in SC$ ,  $s \rightarrow s$ ; and
  - **Anti-symmetric:**  
 $s_1 \rightarrow s_2$  and  $s_2 \rightarrow s_1$  implies  $s_1 = s_2$ ; and
  - **Transitive:**  
 $s_1 \rightarrow s_2$  and  $s_2 \rightarrow s_3$  implies  $s_1 \rightarrow s_3$ .

# Partial orders

- The flow relation  $\rightarrow \subseteq SC \times SC$  is a **partial order**  $(SC, \rightarrow)$  if it is:
  - **Reflexive:**  
for all  $s \in SC$ ,  $s \rightarrow s$ ; and
  - **Anti-symmetric:**  
 $s_1 \rightarrow s_2$  and  $s_2 \rightarrow s_1$  implies  $s_1 = s_2$ ; and
  - **Transitive:**  
 $s_1 \rightarrow s_2$  and  $s_2 \rightarrow s_3$  implies  $s_1 \rightarrow s_3$ .
- Notation for  $\rightarrow$  is then  $\leq$ .

# Partial orders



- The flow relation  $\rightarrow \subseteq SC \times SC$  is a **partial order**  $(SC, \rightarrow)$  if it is:
  - **Reflexive:**  
for all  $s \in SC$ ,  $s \rightarrow s$ ; and
  - **Anti-symmetric:**  
 $s_1 \rightarrow s_2$  and  $s_2 \rightarrow s_1$  implies  $s_1 = s_2$ ; and
  - **Transitive:**  
 $s_1 \rightarrow s_2$  and  $s_2 \rightarrow s_3$  implies  $s_1 \rightarrow s_3$ .
- Notation for  $\rightarrow$  is then  $\leq$ .

# Hasse diagrams

# Hasse diagrams

- **Hasse diagrams** are convenient for representing information flow policies that are partial orders.  
They are directed graphs where:

# Hasse diagrams

- **Hasse diagrams** are convenient for representing information flow policies that are partial orders.  
They are directed graphs where:
  - Security classes are nodes;

# Hasse diagrams

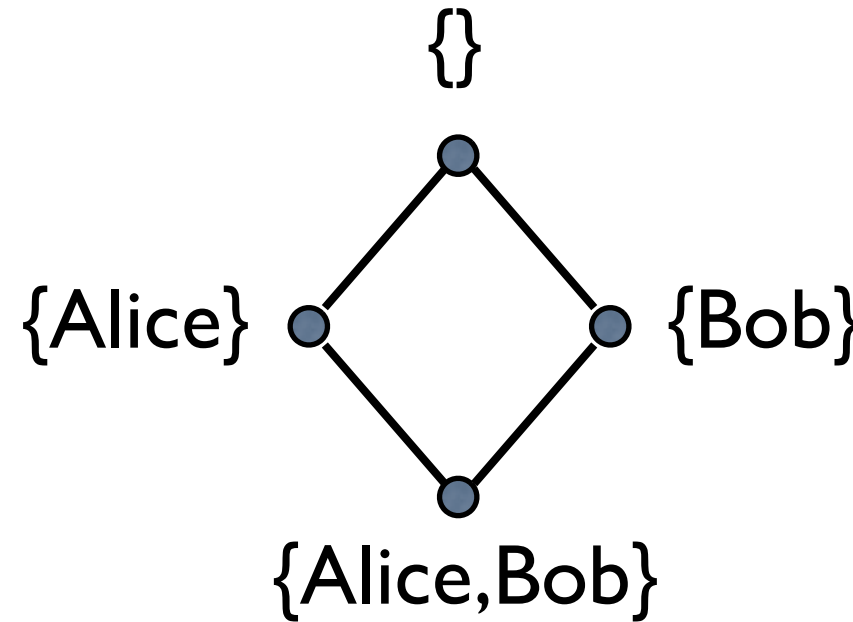
- **Hasse diagrams** are convenient for representing information flow policies that are partial orders.  
They are directed graphs where:
  - Security classes are nodes;
  - The can-flow relation is represented by non-directed arrows, implicitly directed **upward**;



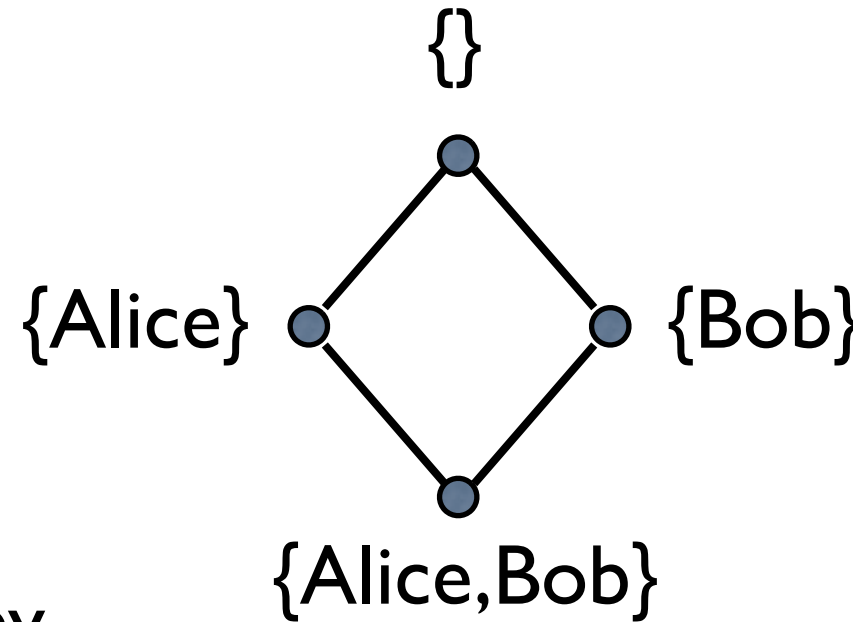
# Hasse diagrams

- **Hasse diagrams** are convenient for representing information flow policies that are partial orders.  
They are directed graphs where:
  - Security classes are nodes;
  - The can-flow relation is represented by non-directed arrows, implicitly directed **upward**;
  - Reflexive and transitive edges are implicit.

# Example: Principal-based policy

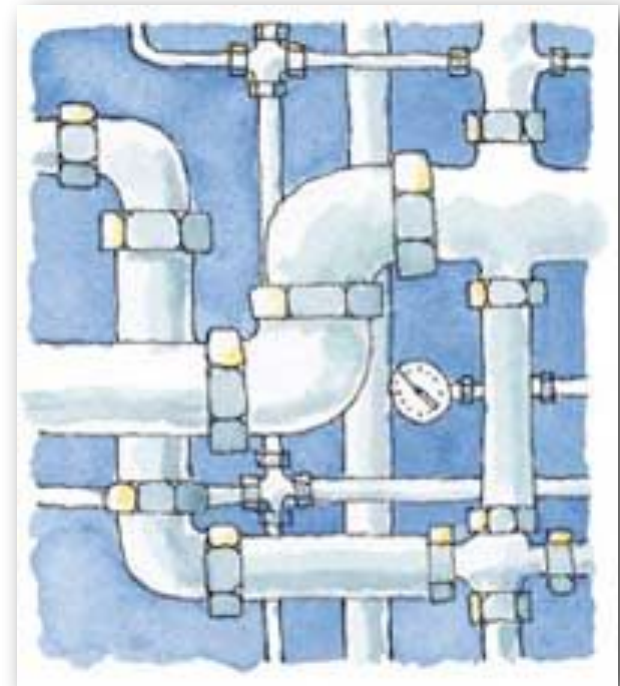


# Example: Principal-based policy



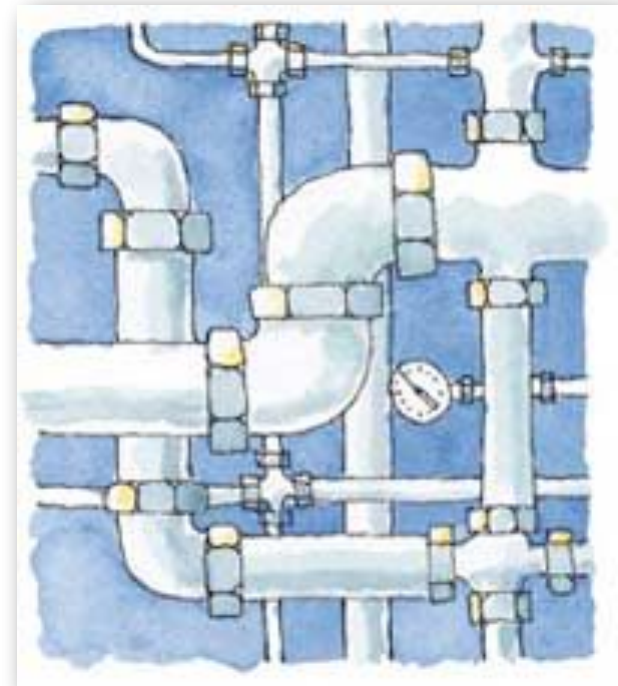
- What is the information flow policy that is represented by the given Hasse diagram?

# “End-to-end” policies



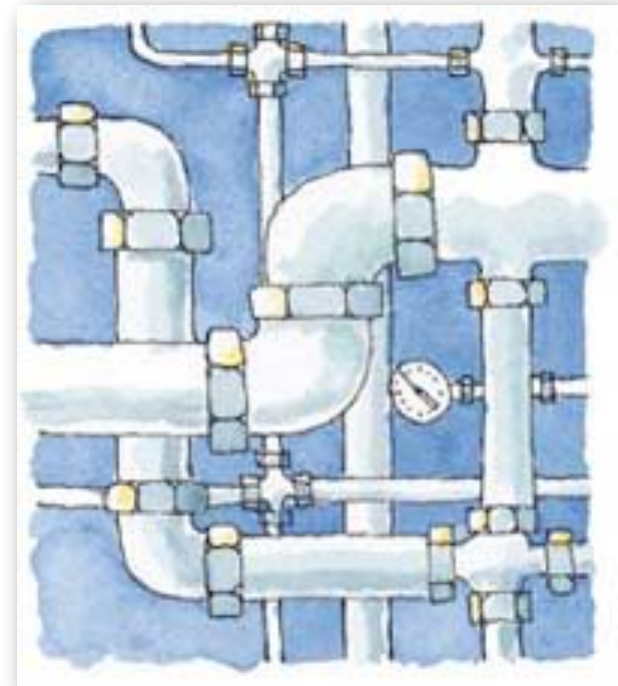
# “End-to-end” policies

- Policies are respected throughout the entire process, as information flows through the system.



# “End-to-end” policies

- Policies are respected throughout the entire process, as information flows through the system.
- Confidentiality / Integrity  
Information of any level should only be readable / affected by users of levels that are the same or higher.





**To think before next class**



# To think before next class

- Can we prevent illegal information flows by means of access control?

# To think before next class

- Can we prevent illegal information flows by means of access control?
- How could we do better?

# Classifying objects

# Classifying objects



# Classifying objects

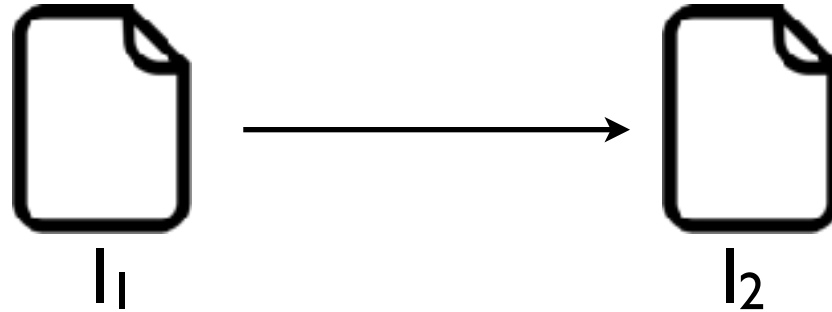


$I_1$

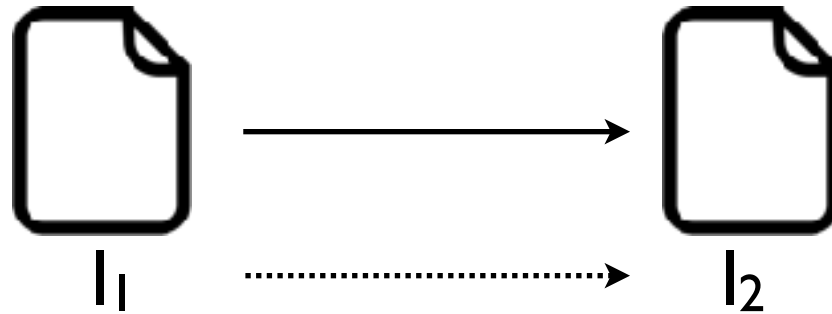


$I_2$

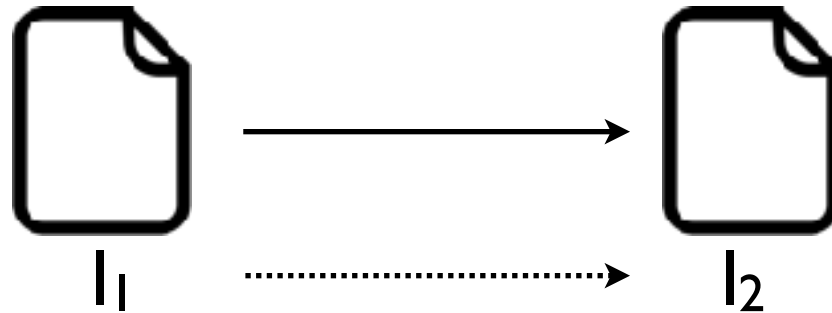
# Classifying objects



# Classifying objects



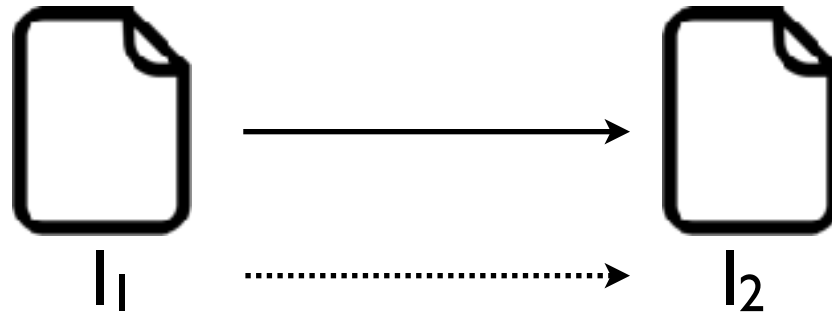
# Classifying objects



- **Object** - resource holding information (ex: variable, file, tuple, channel)

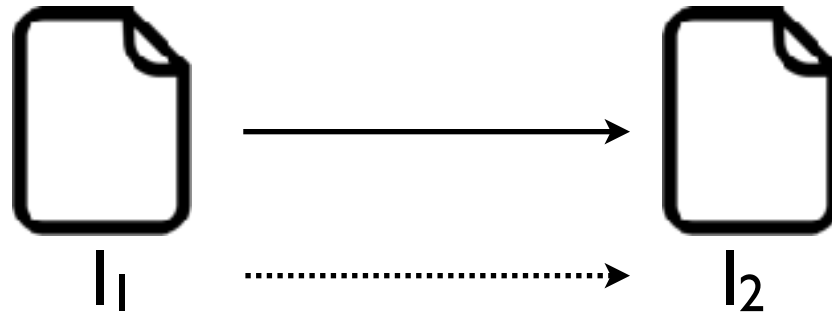


# Classifying objects



- **Object** - resource holding information (ex: variable, file, tuple, channel)
- **Security** class - specifies who can access a certain object

# Classifying objects



- **Object** - resource holding information (ex: variable, file, tuple, channel)
- **Security** class - specifies who can access a certain object
- **Information flow policy** - determines how information can flow between (objects labeled with) security classes.

# What are security classes?

# What are security classes?



# Secure programs?

# Secure programs?

- The attacker: can insert inputs into the system and observe outputs

# Secure programs?

- The attacker: can insert inputs into the system and observe outputs
- We want to ensure that propagation of information by programs respects information flow policies: an attacker cannot infer secret input or affect critical output.

# Secure programs?

- The attacker: can insert inputs into the system and observe outputs
- We want to ensure that propagation of information by programs respects information flow policies: an attacker cannot infer secret input or affect critical output.
- What programs ?



Suppose YOU are the  
attacker

# Suppose YOU are the attacker

- You can only read and write to variables of “low level” -- i.e., your level or lower

# Suppose YOU are the attacker

- You can only read and write to variables of “low level” -- i.e., your level or lower
- For each of the following programs:

# Suppose YOU are the attacker

- You can only read and write to variables of “low level” -- i.e., your level or lower
- For each of the following programs:
  - Can you use the program to uncover “high level” information?

# Suppose YOU are the attacker

- You can only read and write to variables of “low level” -- i.e., your level or lower
- For each of the following programs:
  - Can you use the program to uncover “high level” information?
  - Can you use the program to affect “high level” information?

# Variables as objects (Notation)

H  
|  
L

# Variables as objects (Notation)

- Let us consider a standard imperative language where information containers are variables.

H  
|  
L

# Variables as objects (Notation)

- Let us consider a standard imperative language where information containers are variables.
  - We write  $\mathbf{X}_L$  to denote that a variable  $x$  has security level  $L$ .

H  
|  
L



# Variables as objects (Notation)

- Let us consider a standard imperative language where information containers are variables.
  - We write  $\mathbf{X}_L$  to denote that a variable  $x$  has security level  $L$ .
- We will assume the High-Low lattice



# Preserves confidentiality? <sub>H</sub>

|

you

# Preserves confidentiality?<sub>H</sub>

- $y_{\text{secret}} := x_{\text{you}}$

you

# Preserves confidentiality?<sup>H</sup>

- $y_{\text{secret}} := x_{\text{you}}$  ✓

|  
you

# Preserves confidentiality?<sup>H</sup>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
- $x_{\text{you}} := y_{\text{secret}}$

|  
you

# Preserves confidentiality?<sup>H</sup>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
- $x_{\text{you}} := y_{\text{secret}}$  ✗

|  
you

# Preserves confidentiality?<sup>H</sup>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
- $x_{\text{you}} := y_{\text{secret}}$  ✗



|  
you

# Preserves confidentiality?<sup>H</sup>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
- $x_{\text{you}} := y_{\text{secret}}$  ✗
- if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$



you



# Preserves confidentiality?<sup>H</sup>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
- $x_{\text{you}} := y_{\text{secret}}$  ✗
- if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓



you

# Preserves confidentiality?<sup>H</sup>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
- $x_{\text{you}} := y_{\text{secret}}$  ✗
- if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓
- if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$



you

# Preserves confidentiality?<sup>H</sup>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
- $x_{\text{you}} := y_{\text{secret}}$  ✗
- if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓
- if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  ✗



you

# Preserves confidentiality?<sup>H</sup>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
  - $x_{\text{you}} := y_{\text{secret}}$  ✗
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  ✗
- Explicit leak
- Implicit leak
- you

# Preserves confidentiality?<sup>H</sup>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
  - $x_{\text{you}} := y_{\text{secret}}$  ✗
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  ✗
  - if  $x_{\text{secret}}$  then  $y_H := 0$  else  $y_H := 1$
- Explicit leak
- Implicit leak
- you

# Preserves confidentiality?<sup>H</sup>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
  - $x_{\text{you}} := y_{\text{secret}}$  ✗
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  ✗
  - if  $x_{\text{secret}}$  then  $y_H := 0$  else  $y_H := 1$  ✓
- Explicit leak
- Implicit leak
- you

# Preserves confidentiality?<sup>H</sup>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
  - $x_{\text{you}} := y_{\text{secret}}$  ✗
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  ✗
  - if  $x_{\text{secret}}$  then  $y_H := 0$  else  $y_H := 1$  ✓
  - $x_{\text{low}} := 1$  ; while  $y_{\text{secret}}$  do  $x_{\text{low}} := 0$
- Explicit leak
- Implicit leak
- you

# Preserves confidentiality? <sub>H</sub>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
  - $x_{\text{you}} := y_{\text{secret}}$  ✗
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  ✗
  - if  $x_{\text{secret}}$  then  $y_H := 0$  else  $y_H := 1$  ✓
  - $x_{\text{low}} := 1$  ; while  $y_{\text{secret}}$  do  $x_{\text{low}} := 0$  ✗
- Explicit leak
- Implicit leak
- you



# Preserves confidentiality?<sup>H</sup>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
  - $x_{\text{you}} := y_{\text{secret}}$  ✗
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  ✗
  - if  $x_{\text{secret}}$  then  $y_H := 0$  else  $y_H := 1$  ✓
  - $x_{\text{low}} := 1$  ; while  $y_{\text{secret}}$  do  $x_{\text{low}} := 0$  ✗
- Explicit leak
- Implicit leak
- Implicit leak
- you

# Preserves confidentiality? <sub>H</sub>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
  - $x_{\text{you}} := y_{\text{secret}}$  ✗
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  ✗
  - if  $x_{\text{secret}}$  then  $y_H := 0$  else  $y_H := 1$  ✓
  - $x_{\text{low}} := 1$  ; while  $y_{\text{secret}}$  do  $x_{\text{low}} := 0$  ✗
  - while  $y_{\text{secret}}$  do skip ;  $x_{\text{low}} := 0$
- Explicit leak
- Implicit leak
- Implicit leak
- you

# Preserves confidentiality?<sup>H</sup>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
  - $x_{\text{you}} := y_{\text{secret}}$  ✗
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  ✗
  - if  $x_{\text{secret}}$  then  $y_H := 0$  else  $y_H := 1$  ✓
  - $x_{\text{low}} := 1$  ; while  $y_{\text{secret}}$  do  $x_{\text{low}} := 0$  ✗
  - while  $y_{\text{secret}}$  do skip ;  $x_{\text{low}} := 0$  ?
- Explicit leak
- Implicit leak
- Implicit leak
- you

# Preserves confidentiality? <sub>H</sub>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
  - $x_{\text{you}} := y_{\text{secret}}$  ✗
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  ✗
  - if  $x_{\text{secret}}$  then  $y_H := 0$  else  $y_H := 1$  ✓
  - $x_{\text{low}} := 1$  ; while  $y_{\text{secret}}$  do  $x_{\text{low}} := 0$  ✗
  - while  $y_{\text{secret}}$  do skip ;  $x_{\text{low}} := 0$  ?
  - while  $x_{\text{you}}$  do skip ;  $y_{\text{secret}} := 0$
- Explicit leak
- Implicit leak
- Implicit leak
- you

# Preserves confidentiality? <sub>H</sub>

- $y_{\text{secret}} := x_{\text{you}}$  ✓
  - $x_{\text{you}} := y_{\text{secret}}$  ✗
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓
  - if  $y_{\text{secret}}$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  ✗
  - if  $x_{\text{secret}}$  then  $y_H := 0$  else  $y_H := 1$  ✓
  - $x_{\text{low}} := 1$  ; while  $y_{\text{secret}}$  do  $x_{\text{low}} := 0$  ✗
  - while  $y_{\text{secret}}$  do skip ;  $x_{\text{low}} := 0$  ?
  - while  $x_{\text{you}}$  do skip ;  $y_{\text{secret}} := 0$  ✓
- Explicit leak
- Implicit leak
- Implicit leak
- you

# Preserves integrity?

you

|

H

# Preserves integrity?

you

|

H


- $y_H := x_{\text{you}}$

# Preserves integrity?

you

|

H

- $y_H := x_{\text{you}}$  




# Preserves integrity?

you

|

H



- $y_H := x_{\text{you}}$  
- $x_{\text{you}} := y_H$

# Preserves integrity?

you

|

H


- $y_H := x_{\text{you}}$  
- $x_{\text{you}} := y_H$  


# Preserves integrity?

you

|

H

- $y_H := x_{\text{you}}$  

- $x_{\text{you}} := y_H$  





# Preserves integrity?

you

|

H

- $y_H := x_{\text{you}}$  
- $x_{\text{you}} := y_H$  
- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$




# Preserves integrity?


you


|

H

- $y_H := x_{\text{you}}$  

Explicit leak

- $x_{\text{you}} := y_H$  

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  

# Preserves integrity?

you

|

H

- $y_H := x_{\text{you}}$  ✗

Explicit leak

- $x_{\text{you}} := y_H$  ✓

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$

# Preserves integrity?

you

|

H

- $y_H := x_{\text{you}}$  ✗



- $x_{\text{you}} := y_H$  ✓


- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  ✓

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  ✓


# Preserves integrity?


you


|  
H

- $y_H := x_{\text{you}}$  

← Explicit leak

- $x_{\text{you}} := y_H$  

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  







- if  $x_{\text{you}}$  then  $y_H := 0$  else  $y_H := 1$



# Preserves integrity?

you




H


- $y_H := x_{\text{you}}$   
- $x_{\text{you}} := y_H$  
- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  
- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  
- if  $x_{\text{you}}$  then  $y_H := 0$  else  $y_H := 1$  


# Preserves integrity?

you

H

- $y_H := x_{\text{you}}$   
- $x_{\text{you}} := y_H$  

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  




- if  $x_{\text{you}}$  then  $y_H := 0$  else  $y_H := 1$  


Implicit leak


# Preserves integrity?

you

H

- $y_H := x_{\text{you}}$    Explicit leak
- $x_{\text{you}} := y_H$  

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  

- if  $x_{\text{you}}$  then  $y_H := 0$  else  $y_H := 1$  




 Implicit leak


- while  $y_H$  do skip ;  $x_{\text{low}} := 0$


# Preserves integrity?


you

H

- $y_H := x_{\text{you}}$    Explicit leak
- $x_{\text{you}} := y_H$  

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  

- if  $x_{\text{you}}$  then  $y_H := 0$  else  $y_H := 1$  




- while  $y_H$  do skip ;  $x_{\text{low}} := 0$  


 Implicit leak


# Preserves integrity?

you

H

- $y_H := x_{\text{you}}$    Explicit leak
- $x_{\text{you}} := y_H$  

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  

- if  $x_{\text{you}}$  then  $y_H := 0$  else  $y_H := 1$  

Implicit leak




- while  $y_H$  do skip ;  $x_{\text{low}} := 0$  


- while  $x_{\text{you}}$  do skip ;  $y_H := 0$


# Preserves integrity?

you

H

- $y_H := x_{you}$    Explicit leak
- $x_{you} := y_H$  

- if  $y_H$  then  $x_{you} := 0$  else  $x_{you} := 0$  

- if  $y_H$  then  $x_{you} := 0$  else  $x_{you} := 1$  

- if  $x_{you}$  then  $y_H := 0$  else  $y_H := 1$  

Implicit leak

- while  $y_H$  do skip ;  $x_{low} := 0$  


- while  $x_{you}$  do skip ;  $y_H := 0$  


# Preserves integrity?


you


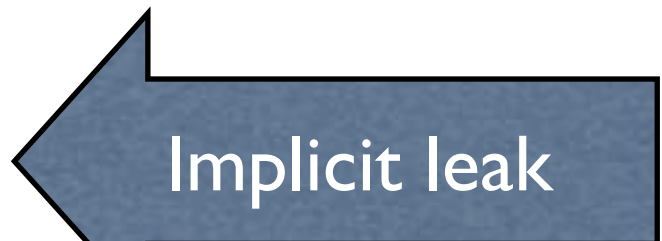
H

- $y_H := x_{\text{you}}$   


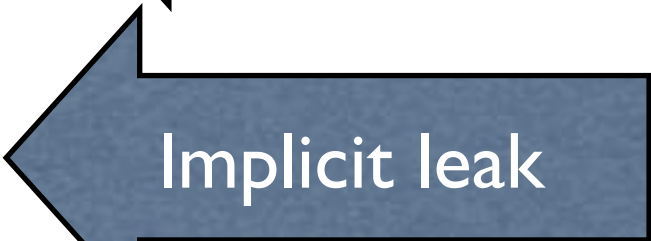
- $x_{\text{you}} := y_H$  

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 0$  

- if  $y_H$  then  $x_{\text{you}} := 0$  else  $x_{\text{you}} := 1$  

- if  $x_{\text{you}}$  then  $y_H := 0$  else  $y_H := 1$   

- while  $y_H$  do skip ;  $x_{\text{low}} := 0$  

- while  $x_{\text{you}}$  do skip ;  $y_H := 0$   

# Conclusion



# Conclusion

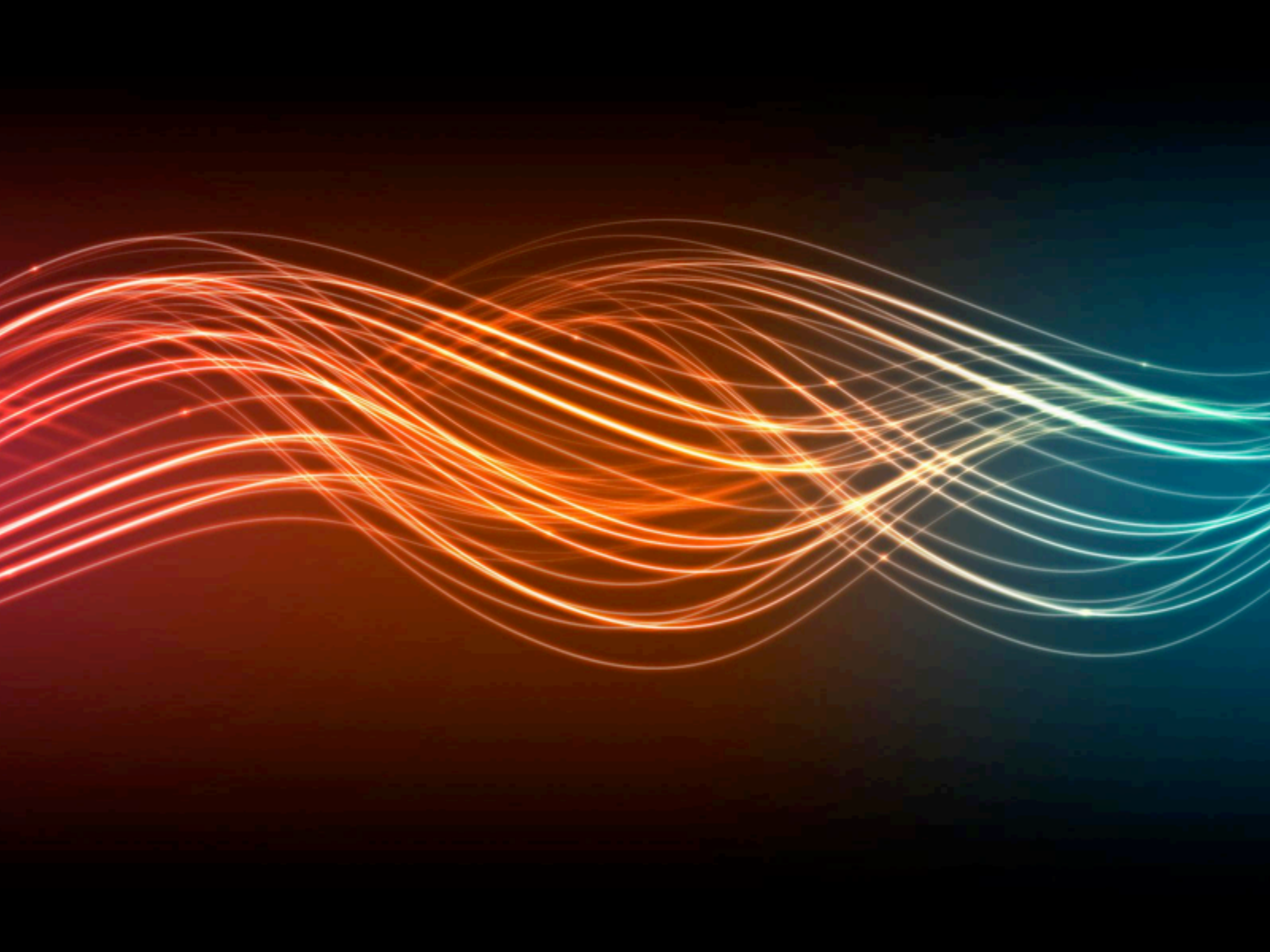
- We want to ensure end-to-end policies such as: an attacker cannot infer secret input or affect critical output by inserting inputs into the system and observing its outputs.

# Conclusion

- We want to ensure end-to-end policies such as: an attacker cannot infer secret input or affect critical output by inserting inputs into the system and observing its outputs.
- these are Information Flow policies.

# Conclusion

- We want to ensure end-to-end policies such as: an attacker cannot infer secret input or affect critical output by inserting inputs into the system and observing its outputs.
- these are Information Flow policies.
- Next class: What are secure programs?



More examples

# Example: Isolated classes

# Example: Isolated classes

- Isolated classes:

# Example: Isolated classes

- Isolated classes:
  - $SC = \{A_1, \dots, A_n\}$



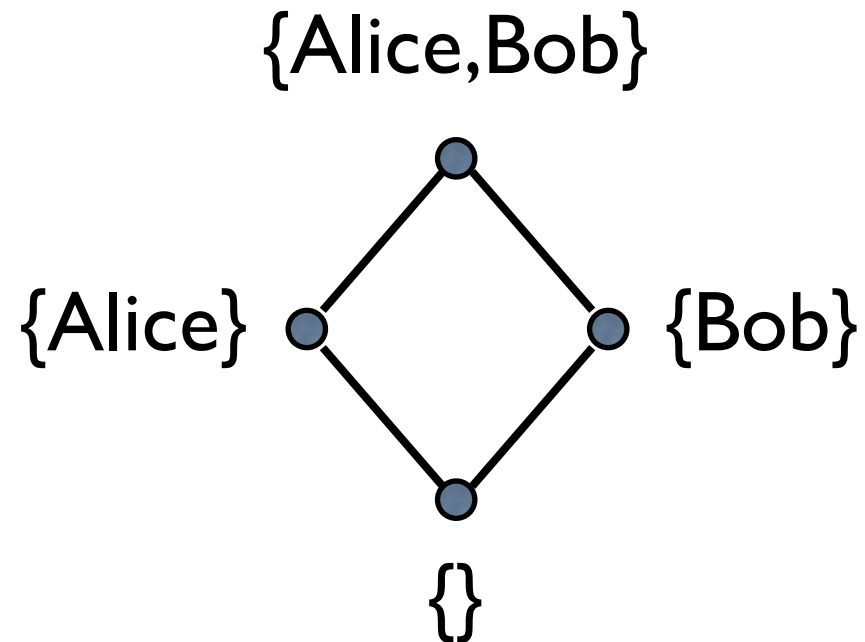
# Example: Isolated classes

- Isolated classes:
  - $SC = \{A_1, \dots, A_n\}$
  - $\rightarrow = \{(A_1, A_1), \dots, (A_n, A_n)\}$   
or, in other words, for all  $A_i \in SC$ ,  $A_i \rightarrow A_i$

# Example: Isolated classes

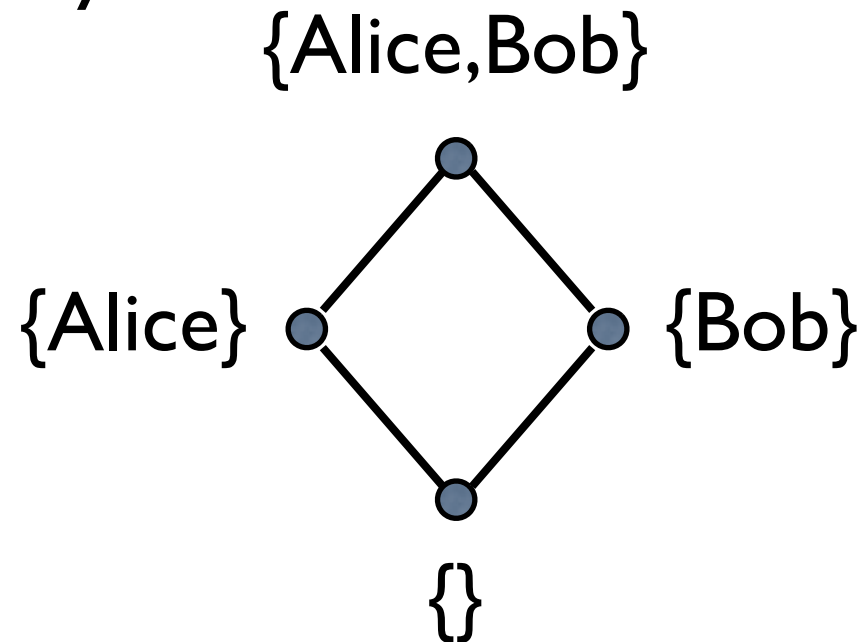
- Isolated classes:
  - $SC = \{A_1, \dots, A_n\}$
  - $\rightarrow = \{(A_1, A_1), \dots, (A_n, A_n)\}$   
or, in other words, for all  $A_i \in SC$ ,  $A_i \rightarrow A_i$
  - $A_1 \oplus A_1 = A_1, \dots, A_n \oplus A_n = A_n$  (otherwise undefined)

# Example: Principal-based policy (integrity)

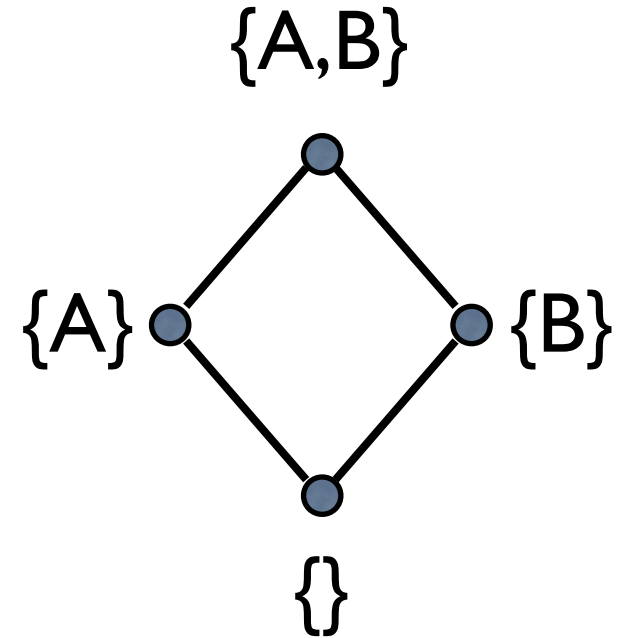


# Example: Principal-based policy (integrity)

- Define the information flow policy that is represented by the following Hasse diagram.

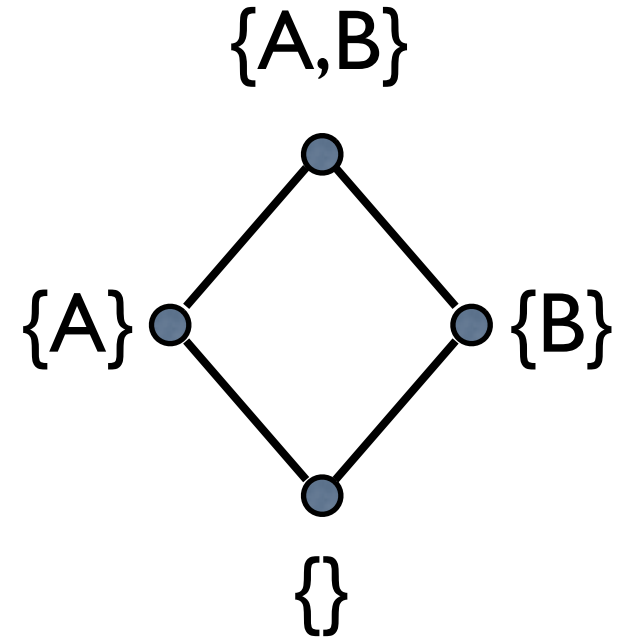


# Example: Principal-based policy



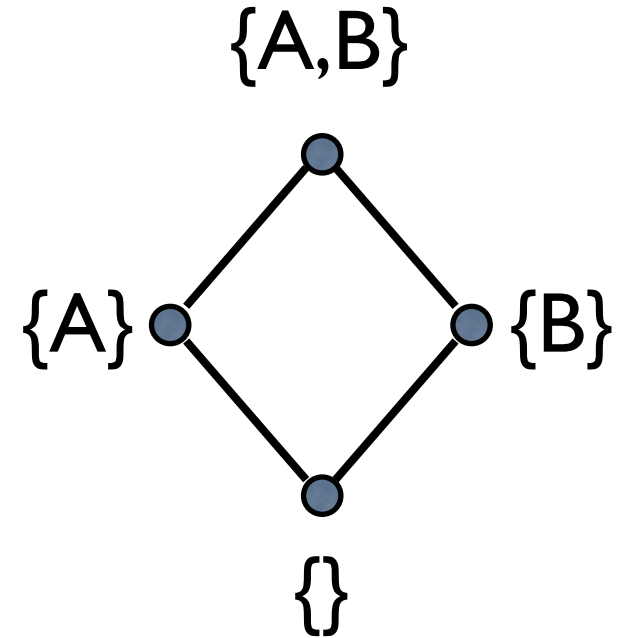
# Example: Principal-based policy

- Principal-based policy for integrity:



# Example: Principal-based policy

- Principal-based policy for integrity:
  - $SC = \{ \{\}, \{A\}, \{B\}, \{A,B\} \}$

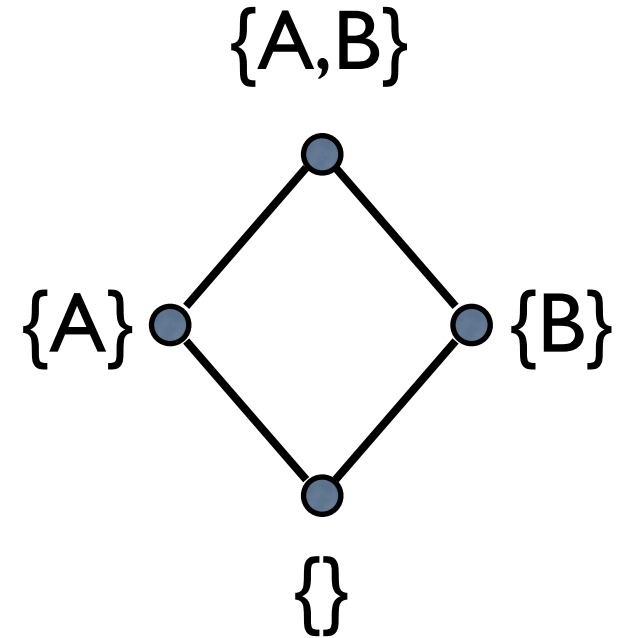


# Example: Principal-based policy

- Principal-based policy for integrity:

- $SC = \{ \{\}, \{A\}, \{B\}, \{A,B\} \}$

- $\rightarrow = \{(\{\}, \{A\}), (\{\}, \{B\}), (\{\}, \{A,B\}),$   
 $(\{A\}, \{A,B\}), (\{B\}, \{A,B\}),$   
 $(\{\}, \{\}), (\{A\}, \{A\}), (\{B\}, \{B\}), (\{A,B\}, \{A,B\})\}$





# Example: Principal-based policy

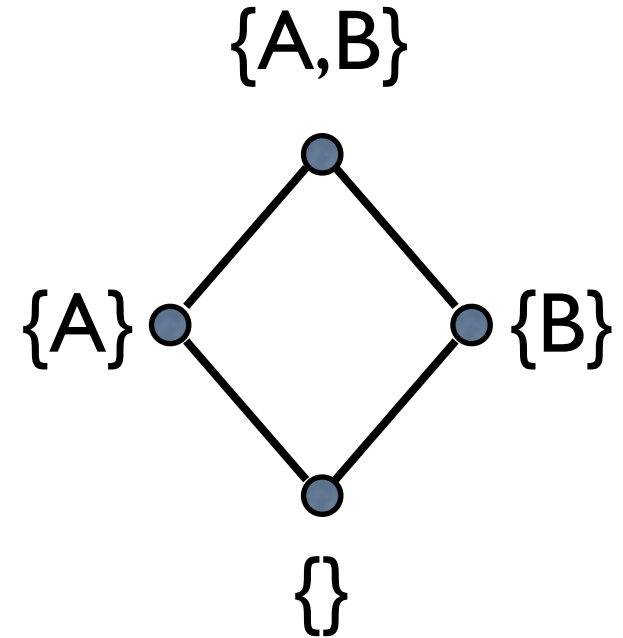
- Principal-based policy for integrity:

- $SC = \{ \{\}, \{A\}, \{B\}, \{A,B\} \}$

- $\rightarrow = \{ (\{\}, \{A\}), (\{\}, \{B\}), (\{\}, \{A,B\}), (\{A\}, \{A,B\}), (\{B\}, \{A,B\}), (\{\}, \{\}), (\{A\}, \{A\}), (\{B\}, \{B\}), (\{A,B\}, \{A,B\}) \}$

(or equivalently)

$$\rightarrow = \subseteq$$



# Example: Principal-based policy

- Principal-based policy for integrity:

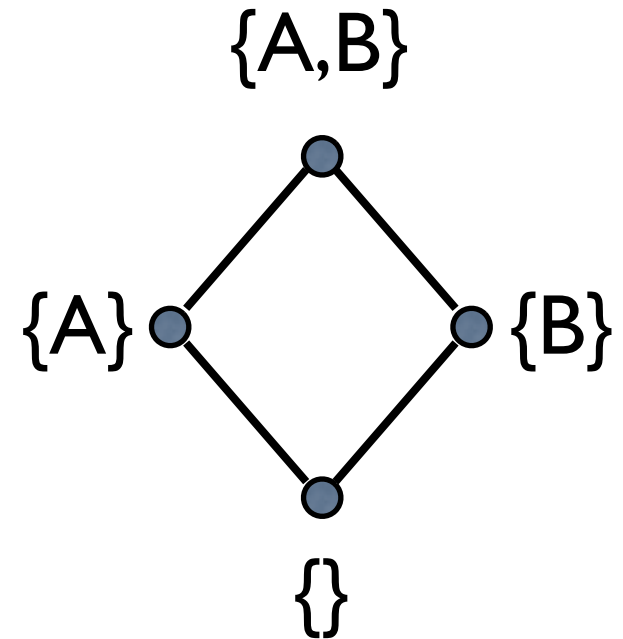
- $SC = \{ \{\}, \{A\}, \{B\}, \{A,B\} \}$

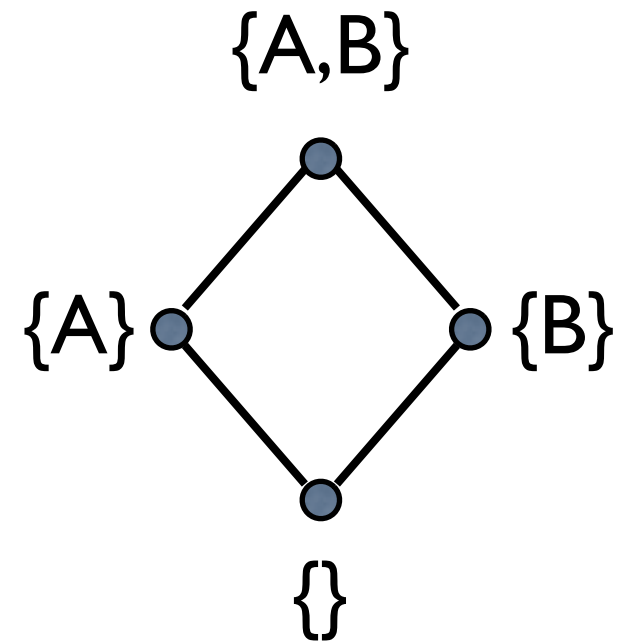
- $\rightarrow = \{ (\{\}, \{A\}), (\{\}, \{B\}), (\{\}, \{A,B\}), (\{A\}, \{A,B\}), (\{B\}, \{A,B\}), (\{\}, \{\}), (\{A\}, \{A\}), (\{B\}, \{B\}), (\{A,B\}, \{A,B\}) \}$

(or equivalently)

$$\rightarrow = \subseteq$$

- ...





- (...)
- $\{\} \oplus \{\} = \{\}, \{A\} \oplus \{A\} = \{A\}, \{B\} \oplus \{B\} = \{B\}, \{A, B\} \oplus \{A, B\} = \{A, B\},$   
 $\{\} \oplus \{A\} = \{A\}, \{\} \oplus \{B\} = \{B\}, \{\} \oplus \{A, B\} = \{A, B\},$   
 $\{A\} \oplus \{A, B\} = \{A, B\}, \{B\} \oplus \{A, B\} = \{A, B\}, \{A\} \oplus \{B\} = \{A, B\}.$   
 (or, equivalently)  
 $\oplus = \cup$