OREGON PROGRAMMING LANGUAGES SUMMER SCHOOL AT BOSTON UNIVERSITY

Harvard John A. Paulson School of Engineering and Applied Sciences

Language-Based Security

Lecture 2: Information Flow Semantics

Stephen Chong, Harvard University

Road Map

- Intro
  - Formal Methods for Security
  - Language-Based Security
  - Case Study: Noninterference
- Primer on Computer Security
- Information Flow
  - Semantics
  - Enforcement
  - Beyond confidentiality
- Enforcing Language Abstractions

Sensitive Information

- Many systems handle a variety of sensitive information
- How do we ensure that the system is handling the information securely?

Access Control Isn't Enough

- Access control can restrict who can *access* information
- But it (typically) doesn't restrict what happens to the information after access
- If "handling information securely" means, e.g., only certain entities should learn about the information, then access control is close, but not exactly aligned

Information Flow

- An **extensional** specification of information security
  - Define security in terms of the observable behavior of the system
  - Not in terms of the implementation details, such as code patterns, mechanisms, etc.
    - i.e., the "intension" of the system
- (Enforcement of an extensional security condition will, of course, depend on implementation details. We will examine enforcement of info flow later.)

Semantics of Information Flow

Strong Dependency

- Cohen (1976) introduced **strong dependency**
  - Essentially, the key definition of noninterference used today
- Intuition: information **flows** from one entity A to another entity B when B depends on or is influenced by A
- Definition: Consider a (deterministic) system **H** whose inputs include entity A and whose outputs include entity B. Output B **strongly depends on** input A if there exist two executions of **H** where the inputs differ only for entity A and the output B differs.
- Security is the absence of certain strong dependencies

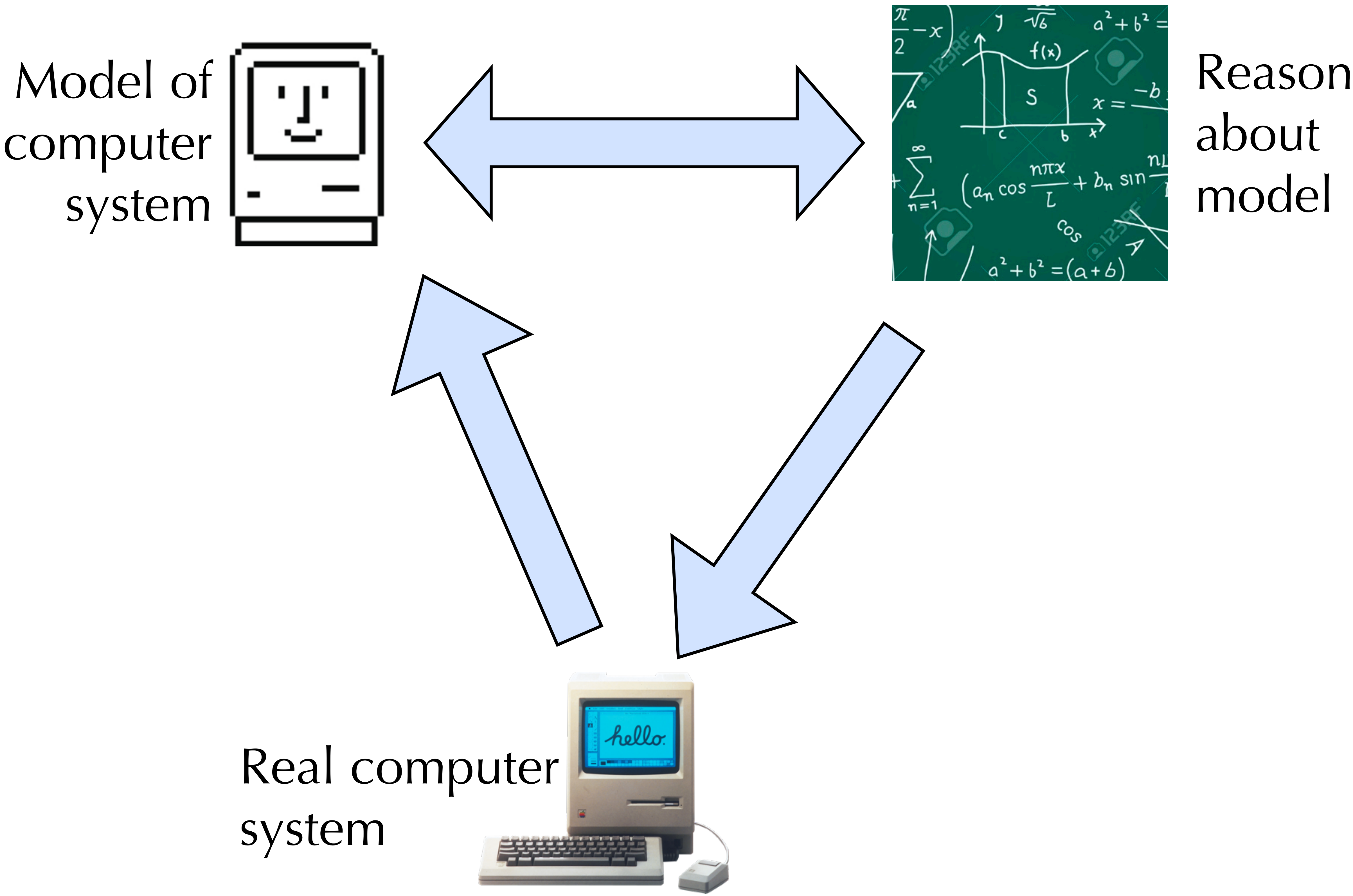
Strong Dependency Example

- In the setting of IMP, with 2 security levels
- Context Γ maps variables to {Low, High}
- Write $\sigma_1 =_{\text{Low}} \sigma_2$ if states σ_1 and σ_2 are equal on all low variables
  - For all x, if $\Gamma(x)$ = Low then $\sigma_1(x) = \sigma_2(x)$
- Definition: Program c is **noninterfering** if:

  For all σ_1, σ_2, σ'_1, σ'_2,

  if $\sigma_1 =_{\text{Low}} \sigma_2$ and $\langle c,\sigma_1\rangle \Downarrow \sigma'_1$ and $\langle c,\sigma_2\rangle \Downarrow \sigma'_2$

  then $\sigma'_1 =_{\text{Low}} \sigma'_2$
- i.e., no strong dependencies from high inputs to low outputs


Model of computer system
Reason about model
Real computer system
hello

Beyond NonInterference for IMP

- In general, formulating an info flow property for a system involves choosing
  - The entities under consideration
    - e.g., who is involved, what's an input, what's an output, ...
  - The conditions under which flows between these entities are allowed or forbidden.
- Much research on info flow over the last 5 decades has considered focused on different *threat models*, *computational models*, and *conditions* that determine whether flows are allowed

Beyond NonInterference for IMP

- **Computational model** indicates the entities that are manipulated during system executions
- **Threat model** indicates the entities with which the adversaries interact.
- Specifying **allowed or forbidden flows** between entities amounts to stipulating allowed or forbidden flows between the system and the adversaries
- We will explore the space of information flow properties by varying the computational model, the threat model, and the expressiveness of the conditions to specify restrictions on info flows

Lecture Roadmap

- Labels and Flow Relations
- Threat model
  - Termination, Timing, and Interaction
  - Computational ability
- Computational model
  - Nondeterminism
  - Probability
  - Concurrency
- Reclassification
  - Quantitative info flow

Labels

- Syntactic objects associated with entities of a system
  - E.g., *Secret*, *Public*
  - E.g., *Trusted*, *Untrusted*
  - E.g., *Alice*, *Bob*, *Charlie*, ...
  - E.g., (Level, Compartment) where
    Level ∈ { *Public*, *Confidential*, *Secret*, *TopSecret*} and
    Compartment ∈ { *Nuclear*, *Cryptography*, *Biological*, ...}
- Info-flow policy might described allowed (or forbidden) flows between entities based on labels
- Labels might have rich structure but don't themselves describe policies
  - Labels represent restrictions on how associated entities can be used

Flow Relations

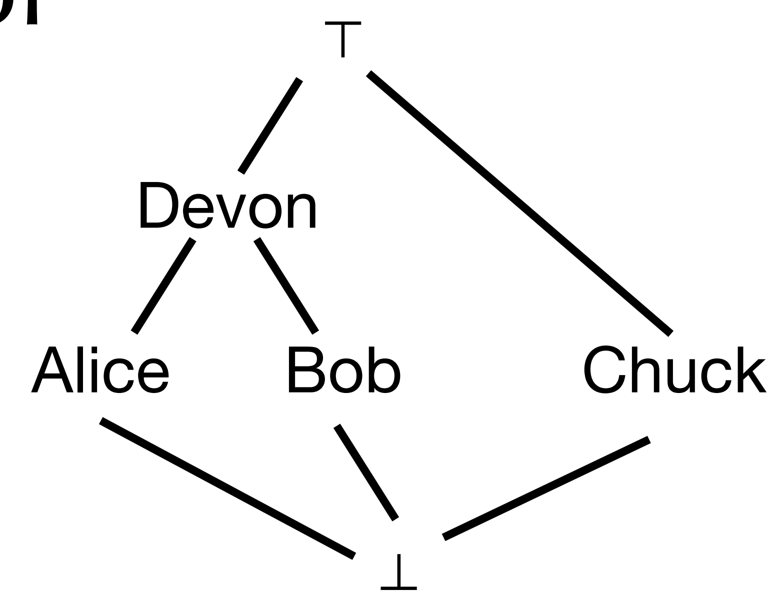
- Info flow policy often represented as flow relation $\sqsubseteq$ on a set Λ of labels
  - If $\ell_1 \sqsubseteq \ell_2$ then info is allowed to flow from ℓ_1 to ℓ_2
- What structure should flow relation $\sqsubseteq$ have?
  - Reflexive, i.e., for all $\ell \in \Lambda$ we have $\ell \sqsubseteq \ell$
  - Transitive?
    - i.e., for all $\ell_1, \ell_2, \ell_3 \in \Lambda$, if $\ell_1 \sqsubseteq \ell_2$ and $\ell_2 \sqsubseteq \ell_3$ then $\ell_1 \sqsubseteq \ell_3$
  - Reflexive and transitive is a **pre-order**
  - If we add antisymmetry, it is a **partial order**

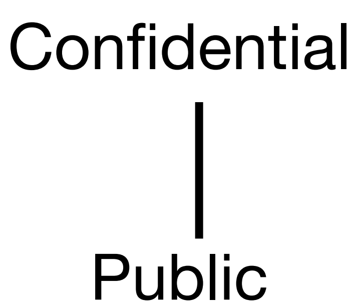
Lattice

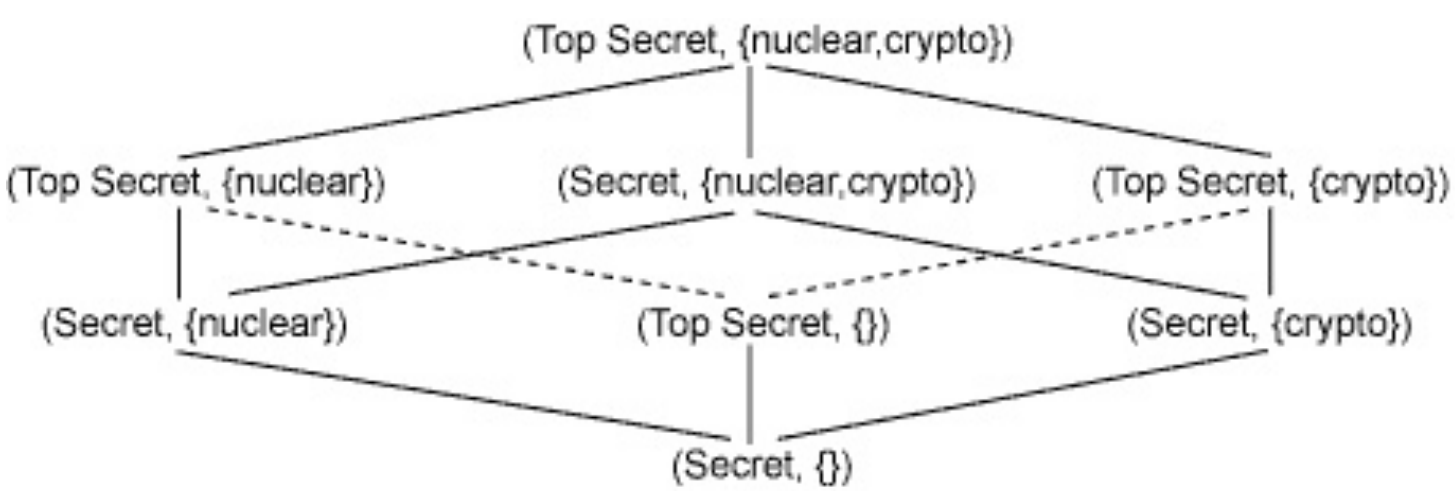
- Denning (1978) argues for a join-semi-lattice relation
  - i.e., a **least-upper bound** operation $\sqcup$
    - Upper bound: $\forall\ \ell_1, \ell_2 \in \Lambda,\ \ell_1 \sqsubseteq \ell_1 \sqcup \ell_2$ and $\ell_2 \sqsubseteq \ell_1 \sqcup \ell_2$
    - Least upper bound: $\forall\ \ell_1, \ell_2, \ell_3 \in \Lambda$, if $\ell_1 \sqsubseteq \ell_3$ and $\ell_2 \sqsubseteq \ell_3$ then $\ell_1 \sqcup \ell_2 \sqsubseteq \ell_3$
- Why?
- Given data a and b, labeled respectively ℓ_a and ℓ_b
- What should be label of operation $a \oplus b$?
  - Should be upper bound
  - Should be least upper bound, otherwise the following may not work (where ℓ_{d1} and ℓ_{d2} are both upper bounds of ℓ_a and ℓ_b)
    - $c = a \oplus b;\ d1 = c;\ d2 = c$

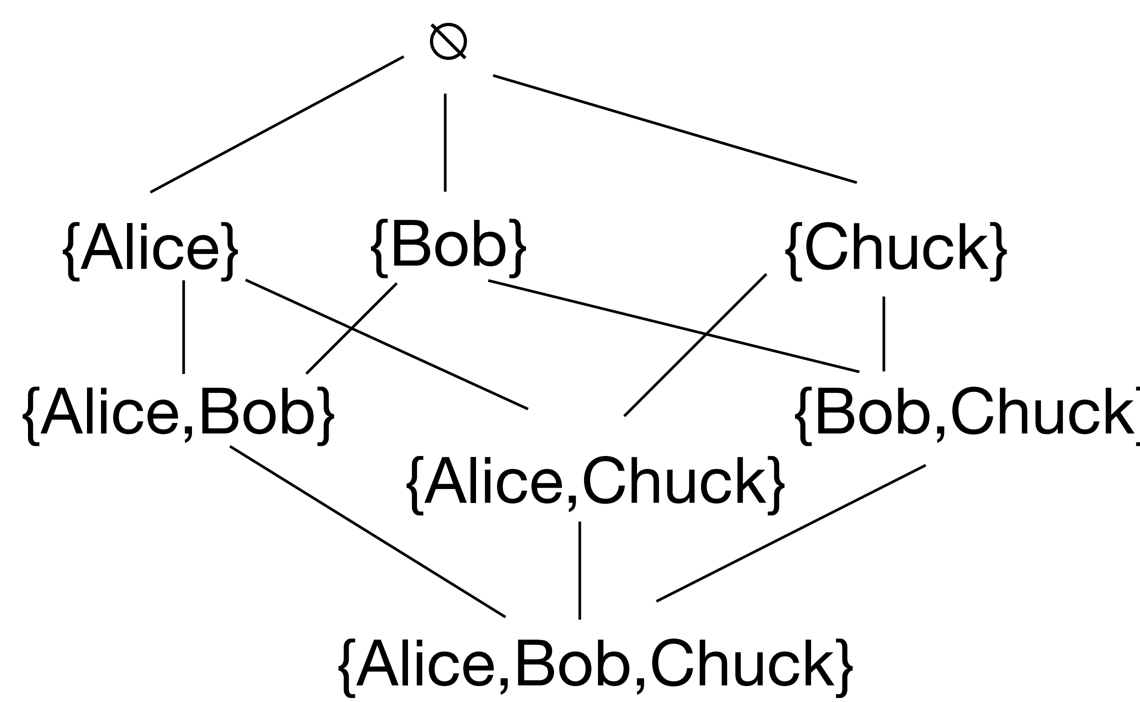
From Labels to NI

- Here is a more general version of noninterference:
  - Lattice $(\Lambda, \sqsubseteq)$ of security levels









From Labels to NI

- Here is a more general version of noninterference:
  - Lattice $(\Lambda, \sqsubseteq)$ of security levels
  - Context Γ is function from variables to Λ
  - Write $\sigma_1 =_\ell \sigma_2$ if states σ_1 and σ_2 are equal on all low variables: For all x, if $\Gamma(x) \sqsubseteq \ell$ then $\sigma_1(x) = \sigma_2(x)$
  - Definition: Program c is **noninterfering** if:

    For all $\sigma_1, \sigma_2, \sigma'_1, \sigma'_2, \ell \in \Lambda$

    if $\sigma_1 =_\ell \sigma_2$ and $\langle c,\sigma_1\rangle \Downarrow \sigma'_1$ and $\langle c,\sigma_2\rangle \Downarrow \sigma'_2$

    then $\sigma'_1 =_\ell \sigma'_2$

Lecture Roadmap

- Labels and Flow Relations
- Threat model
  - Termination, Timing, and Interaction
  - Computational ability
- Computational model
  - Nondeterminism
  - Probability
  - Concurrency
- Reclassification
  - Quantitative info flow

Threat Model

- How adversary interacts with system
- Stronger threat model → more interactions → more opportunities for information flow to/from adversary
- **Information channels** convey information
  - Lampson (1973) categorizes them as:
  - **legitimate channels** (e.g., files, console, network messages, ...) and
  - **covert channels** (e.g., execution time, heat emission, noise emission, resource exhaustion, power consumption, ...)
    - **side channels** are covert channels exploited by passive adversary who simply observes the channel

Termination

- Earlier definition of NI is **termination-insensitive**
  - Implicitly assumes that attacker ignores all executions that fail to terminate

> - Definition: Program c is **noninterfering** if:
>   For all σ_1, σ_2, σ'_1, σ'_2, $\ell \in \Lambda$
>   if $\sigma_1 =_\ell \sigma_2$ and $\langle c, \sigma_1 \rangle \Downarrow \sigma'_1$ and $\langle c, \sigma_2 \rangle \Downarrow \sigma'_2$
>   then $\sigma'_1 =_\ell \sigma'_2$

  - So `while (high > 0) do skip` satisfies noninterference

Termination-Sensitivity

- Can modify security condition to account for termination channel
  - Key idea: termination behavior is determined by low inputs
    - Either both executions terminate or both executions diverge
- Definition: Program c is **termination-sensitive noninterfering** if:

  For all σ_1, σ_2, ℓ, if $\sigma_1 =_\ell \sigma_2$ then either

  - exists σ'_1, σ'_2. $\langle c,\sigma_1\rangle \Downarrow \sigma'_1$ and $\langle c,\sigma_2\rangle \Downarrow \sigma'_2$ and $\sigma'_1 =_\ell \sigma'_2$

  or

  - both executions diverge

Time

- Can the adversary observe how long an execution takes?
  - **Timing sensitivity**
- Termination sensitivity is an extreme example of timing sensitivity
- Several ways of thinking about timing
  - Number of steps the computational model takes
    - But suffers from big gap between model and reality
  - External timing ("Wall clock time")
    - Actually very hard to capture accurately in a model, as it depends on many low-level system details
      - Memory hierarchy, microarchitecture details, …

- Conceptually, can add new variable to state, T, which increases during execution and is low-observable

  - Internal timing
    - E.g., thread running in the same system that can detect which event happens first

- Concurrency (see later)

Interaction

- So far we have used a “batch”-like model of computation
  - Systems gets input, does all computation and produces output on termination
- Most systems are interactive
  - Adversary may make observations during executions
  - Adversary (and others) may provide inputs during execution
- Requires different computational model to express

Interaction

- Assume IMP with `x := input from` ℓ and `output x to` ℓ
- Semantics $\langle c,\sigma\rangle \longrightarrow^{\tau} \langle c',\sigma'\rangle$ where **trace** τ is a sequence of events
  - $\tau ::= \varepsilon \quad | \quad \tau \cdot \mathrm{in}(n, \ell) \quad | \quad \tau \cdot \mathrm{out}(n, \ell)$
  - Intuitively: $\langle c,\sigma\rangle$ takes one or more steps to $\langle c',\sigma'\rangle$ producing trace τ
- Interactive noninterference: if initial memories are low equivalent and low inputs are identical, then the traces are low-equivalent (i.e., low inputs and outputs are the same)
- Definition: Program c is **noninterfering** if:

  For all σ_1, σ_2, σ'_1, σ'_2, τ_1, τ_2, $\ell \in \Lambda$

  if $\langle c,\sigma_1\rangle \longrightarrow^{\tau_1} \langle \mathrm{skip}, \sigma'_1\rangle$ and $\langle c,\sigma_2\rangle \longrightarrow^{\tau_2} \langle \mathrm{skip}, \sigma'_2\rangle$

  and $\sigma_1 =_\ell \sigma_2$ and $\mathrm{inputs}(\tau_1) =_\ell \mathrm{inputs}(\tau_2)$

  then $\tau_1 =_\ell \tau_2$

Progress Sensitivity

- Can the attacker observe whether program is making progress (i.e., will produce another event)?
- Analogous to termination sensitivity, but for non-batch programs

Program Code

- Does the attacker know the code? Can they modify/ provide code?
- Noninterference typically (implicitly) assumes attacker knows code

> - Definition: Program c is **noninterfering** if:
>   For all σ_1, σ_2, σ'_1, σ'_2, ℓ
>   if $\sigma_1 =_\ell \sigma_2$ and $\langle c,\sigma_1\rangle \Downarrow \sigma'_1$ and $\langle c,\sigma_2\rangle \Downarrow \sigma'_2$
>   then $\sigma'_1 =_\ell \sigma'_2$

- Some models allow attacker to provide code (but this can typically be simulated by any attacker-provided input)

Attacker's Computational Ability

- What can the attacker compute?
- E.g., does the following satisfy noninterference?

```
Γ(msg1) = Low
Γ(msg2) = Γ(key) = High
output encrypt(msg1, key) to Low
output msg1 to Low
output encrypt(msg2, key) to Low
```

- Some versions of noninterference assume computational limits on attacker

Views of a System

- More generally, may define what the attacker can observe as a **view** of the system, a function from the system state (or history) to the attacker's observations
- E.g., attacker sees a subset of the state of the system
  - Appropriate for a distributed system where some machines are compromised
- E.g., attacker sees power consumption of system
- Definition: Program c is **noninterfering** if:

  For all σ_1, σ_2, τ_1, τ_2, $\ell \in \Lambda$

  if $\sigma_1 =_\ell \sigma_2$ and $\langle c,\sigma_1\rangle \Downarrow \tau_1$ and $\langle c,\sigma_2\rangle \Downarrow \tau_2$

  then $\mathrm{view}(\ell, \tau_1) = \mathrm{view}(\ell, \tau_2)$

  - (Haven't defined relation $\langle c,\sigma\rangle \Downarrow \tau$. Think of τ of as being the history of the computation, includes events, states, ..., anything we want to model as observable)

Threat Model Summary

- Many different versions of non-interference handle different threat models
- From Kozyri et al.

The adversary can:	**Example security conditions**
Observe termination	Termination-sensitive noninterference
Observe time	Time-sensitive noninterference
Observe output stream	Progress-sensitive noninterference
and provide input stream	Reactive noninterference, GMNI, non-inference, generalized noninference
and use input strategies	Nondeducibility on strategies
and be a concurrently executed program	P_BNDC
Write program code	Noninterference against active adversary
Observe views of system behavior	Nondeducibility, Opaqueness

Lecture Roadmap

- Labels and Flow Relations
- Threat model
  - Termination, Timing, and Interaction
  - Computational ability
- Computational model
  - Nondeterminism
  - Probability
  - Concurrency
- Reclassification
  - Quantitative info flow

Computational Model

- Computational model abstracts system functionality
  - Tightly coupled with threat model
- Computational model captures implementation details of a system, at varying levels of faithfulness

Nondeterminism

- So far we have considered deterministic systems
- Noninterference doesn't hold for nondeterministic system
- E.g., with nondeterministic choice operator $a_1 \square a_2$, program

```
low := 42 □ 7
```

  may not satisfy NI
- Intuitively, we don't know how nondeterminism is resolved; may depend on secret information
  - So-called **refinement attack**

Generalized Noninterference

- Intuition: secret inputs do not constrain public outputs
  - i.e., all possible Low behaviors are possible with any High inputs
- Definition: Program c satisfies

  generalized noninterference if:

  For all σ_1, σ_2, σ'_1, σ'_2,

  if $\sigma_1 =_{\text{Low}} \sigma_2$ and $\langle c,\sigma_1\rangle \Downarrow \sigma'_1$ and $\langle c,\sigma_2\rangle \Downarrow \sigma'_2$

  then there exists σ_3, σ'_3 such that

  $\sigma_3 =_{\text{Low}} \sigma_1$ and $\sigma_3 =_{\text{High}} \sigma_2$ and

  $\langle c,\sigma_3\rangle \Downarrow \sigma'_3$ and $\sigma'_3 =_{\text{Low}} \sigma'_1$

Observational Determinism

- But resolution of nondeterminism is useful!
- **Observational determinism** requires that resolution of Low nondeterminism does not depend on secret information
  - E.g., if nondeterminism is due to scheduler choices of threads/ processes, the scheduler should not depend on high information
- Definition same as deterministic NI! i.e., low view is determined by low inputs
  - For all σ_1, σ_2, τ_1, τ_2, $\ell \in \Lambda$
    if $\sigma_1 =_\ell \sigma_2$ and $\langle c,\sigma_1\rangle \Downarrow \tau_1$ and $\langle c,\sigma_2\rangle \Downarrow \tau_2$
    then $\text{view}(\ell, \tau_1) = \text{view}(\ell, \tau_2)$
- Pro: not subject to refinement attack
- Con: allows no public nondeterminism

Probability

- **Possibilistic** nondeterminism may not sufficiently model information flows if some choices are unlikely
  - E.g.,

```
a := 0 □ 1 □ ... □ 999;
if (a = 0) then low := 0 □ 1
else low := high mod 2
```

- **Probabilistic** noninterference requires that the ***distribution*** of low outputs is independent of high inputs
- Assume probabilistic semantics $\langle c,\sigma\rangle \Downarrow \mathfrak{D}$ where $\mathfrak{D}$ is a (sub-)distribution over stores
  - Add your favorite probabilistic operators to the language
- Probabilistic Noninterference: For all σ_1, σ_2, $\mathfrak{D}_1$, $\mathfrak{D}_2$, $\ell \in \Lambda$

  if $\sigma_1 =_\ell \sigma_2$ and $\langle c,\sigma_1\rangle \Downarrow \mathfrak{D}_1$ and $\langle c,\sigma_2\rangle \Downarrow \mathfrak{D}_2$

  then $\mathfrak{D}_1|_\ell = \mathfrak{D}_2|_\ell$
  - (Where $\mathfrak{D}|_\ell$ projects the distribution over stores to a distribution over the low-observable part of the store)

Concurrency

- When modeling concurrency, information might flow by
  - Interaction between threads
    - E.g., race conditions are a source of nondeterminism
  - Scheduling choices
  - Memory model
    - Sequential consistency, Total Store Order, Partial Store Order, ...
- Relatedly, speculative execution is source of real information leaks
  - E.g., Spectre and Meltdown attacks
- Don't really need a new definition of noninterference
  - Other than extending our language and semantics to support concurrency

Lecture Roadmap

- Labels and Flow Relations
- Threat model
  - Termination, Timing, and Interaction
  - Computational ability
- Computational model
  - Nondeterminism
  - Probability
  - Concurrency
- Reclassification
  - Quantitative info flow

Reclassification

- In practice, noninterference is too restrictive
  - Information does not keep the same label throughout execution
- May need to **declassify** information
  - i.e., weaken confidentiality requirements
  - e.g., credit card number is confidential, but last 4 digits can be printed on receipt
  - e.g., when a physician is assigned to a patient, they can see the patient's records
  - e.g., after a sealed-bid auction is concluded, the confidential bids may be made publics
- May need to **erase** information
  - i.e., strengthen confidentiality requirements
  - e.g., after transaction, merchant should no longer hold credit card information
  - e.g., when submarine surfaces, sensitive information should be encrypted

Handling Information Appropriately

- How to declassify in a controlled way?
  - Don't want to allow all confidential information to be released!
- Sabelfeld and Sands (2009) describe "dimensions" of declassification:
  - *What* info is declassified
  - *Who* declassifies the info
  - *Where* in the system (i.e., component) or label relation does the declassification
  - *When* (under what conditions) does declassification happen?

Example: Escape Hatches

- Delimited Release (Sabelfeld and Myers, 2003)
- Intuition: specifies *what* information may be declassified by a set of **escape hatch** expressions
- Definition: Program c and set of escape hatches $\{a_1, \ldots, a_n\}$ satisfies **delimited release** if:

  For all σ_1, σ_2, σ'_1, σ'_2,

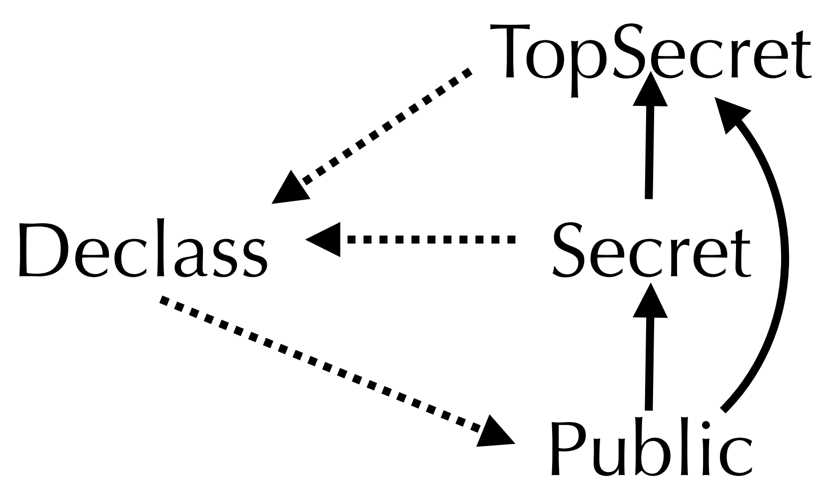
  if $\sigma_1 =_{Low} \sigma_2$ and $\langle c,\sigma_1\rangle \Downarrow \sigma'_1$ and $\langle c,\sigma_2\rangle \Downarrow \sigma'_2$

  and for all $i \in 1..n$, $\sigma_1(a_i) = \sigma_2(a_i)$

  then $\sigma'_1 =_{Low} \sigma'_2$

Example: Intransitive NI

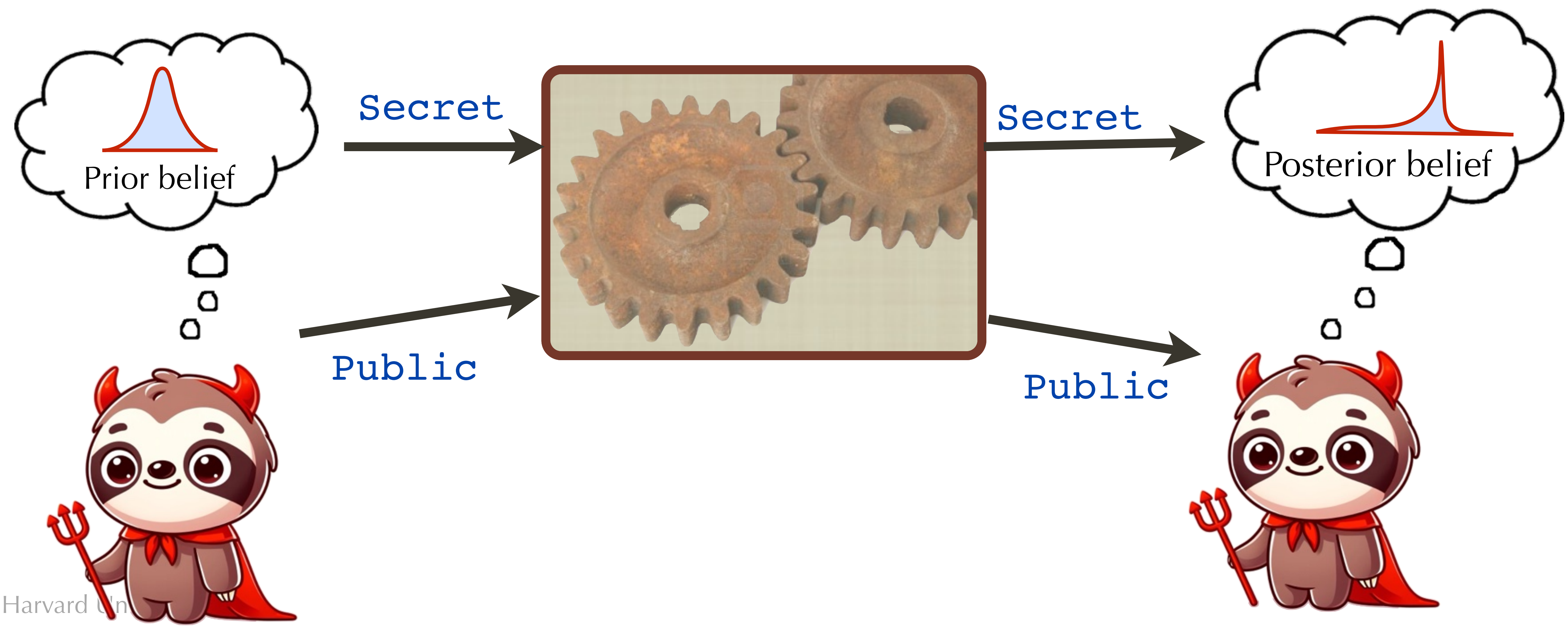
- Intuition: remove transitivity as a requirement for the flow relation
- E.g.,



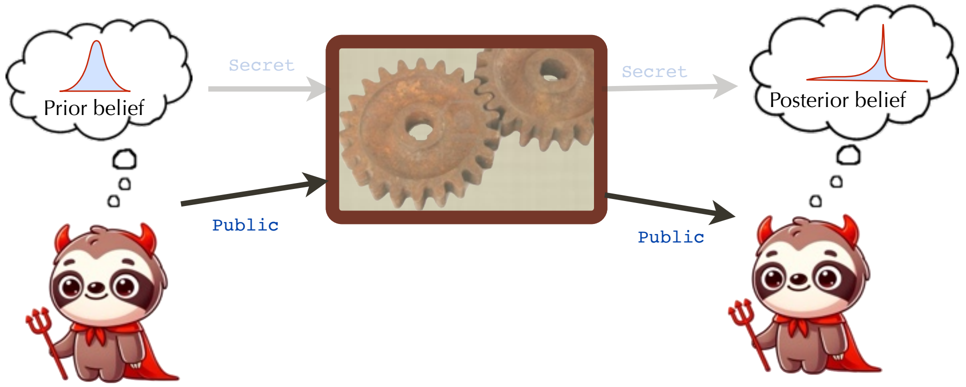
- Typically a **trusted component** is the only component that is permitted to use intransitive flow relations (a form of *where* declassification)
- Security conditions might need to consider some of the implementation details to express this...

Quantitative Info Flow

- There are info leaks that are undesirable but unavoidable (e.g., via side channels)
- How to understand the magnitude of these leaks?
- **Quantitative information flow** uses information theory to measure leakage
- Basic idea: info leakage = initial uncertainty – remaining uncertainty



Quantitative Info Flow



- Different ways of measuring leakage, e.g.,
  - Shannon entropy
  - Bayes vulnerability
  - Renyi's min-entropy
  - Not all bits are equal: gain functions can capture value of bits
- E.g., Shannon entropy
  - For random variable X, H(X) is the **Shannon entropy** of X
    - Expected number of bits to optimally encode value of X
    - Uncertainty or surprise in X

$$H(X) = -\sum_{x \in \mathcal{X}} \Pr(x) \log_2 (\Pr(x))$$

  - Conditional entropy H(X|Y) information in X given knowledge of Y

$$H(X|Y) = \sum_{y \in \mathcal{Y}} \Pr(Y = y) H(X|Y = y)$$

- Leakage = $H(\mathrm{In}_{\mathrm{Secret}}) - H(\mathrm{In}_{\mathrm{Secret}} \mid \mathrm{In}_{\mathrm{Public}}, \mathrm{Out}_{\mathrm{Public}})$

Selected References

- Kozyri, E., S. Chong, and A. C. Myers (2022). Expressing information flow properties. Foundations and Trends in Privacy and Security 3(1), 1–102.
- Sabelfeld, A. and A. C. Myers (2003, January). Language-based information-flow security. IEEE Journal on Selected Areas in Communications 21(1), 5–19.
- Sabelfeld, A. and D. Sands (2005, June). Dimensions and principles of declassification. In Proceedings of the 18th IEEE Computer Security Foundations Workshop, pp. 255–269. IEEE Computer Society.
- Alvim, M. S., K. Chatzikokolakis, A. McIver, C. Morgan, C. Palamidessi, and G. Smith (2020). The Science of Quantitative Information Flow. Springer.