Seguridad Informatica - Fall 2024

Module IV: Physical security

Lecture 4

Non-interference

Marco Guarnieri
IMDEA Software Institute

Module IV

- Lecture 1 — Introduction
- Lecture 2 — Cache-based side channel attacks
- Lecture 3 — Speculative execution attacks
- Lecture 4 — Non-interference
- Lecture 5 — Automated detection of speculative leaks

Module IV

- Lecture 1 — Introduction
- Lecture 2 — Cache-based side channel attacks
- Lecture 3 — Speculative execution attacks
- Lecture 4 — Non-interference
- Lecture 5 — Automated detection of speculative leaks

Recommended reading

Language-based information-flow basics

by Aslan Askarov

Available at https://github.com/aslanix/SmallStepNI/blob/master/infoflow-basics.pdf

Introduction

- ***Problem:*** Seemingly secure programs can leak sensitive information

Introduction

- ***Problem:*** Seemingly secure programs can leak sensitive information
- The ***question*** that we will answer:
  - When is a program ***secure***?

Introduction

- ***Problem:*** Seemingly secure programs can leak sensitive information
- The ***question*** that we will answer:
  - When is a program ***secure***?

Security as ***confidentiality***!

Introduction

- ***Problem:*** Seemingly secure programs can leak sensitive information
- The ***question*** that we will answer:
  - When is a program ***secure***?

Security as ***confidentiality***!

- How can we ***reason*** about a program security?
  - ***Non-interference*** provides theoretical foundations for talking about ***program security***

What is the meaning of a program?

Programs

Programs map ***memories*** to ***memories***

$$\langle\langle Prg \rangle\rangle : Mems \rightarrow Mems$$

Programs

Programs map ***memories*** to ***memories***

$$\langle\!\langle Prg \rangle\!\rangle : Mems \rightarrow Mems$$

Program

Programs

Programs map ***memories*** to ***memories***

$$\langle\!\langle Prg \rangle\!\rangle : Mems \to Mems$$

Program

Program memories: map variables to values

Programs

Programs map ***memories*** to ***memories***

Examples

$$\langle\!\langle x \leftarrow x + 1 \rangle\!\rangle([x \mapsto 0]) = [x \mapsto 1]$$

$$\langle\!\langle x \leftarrow x + 1 \rangle\!\rangle([x \mapsto 1, y \mapsto 2]) = [x \mapsto 2, y \mapsto 2]$$

Program

Program memories: map variables to values

Formalizing programs - *Syntax*

$$x \in Vars$$

$$n \in \mathbb{N}$$

$$\ominus \in \{\neg, \ldots\}$$

$$\oplus \in \{+, -, *, \ldots\}$$

$$expr = n \mid x \mid \ominus\ expr \mid expr_1\ \oplus\ expr_2$$

$$stmt = \texttt{stop} \mid \texttt{skip} \mid x \leftarrow expr \mid stmt_1; stmt_2$$

$$\mid \texttt{if}(expr)\{stmt_1\}\texttt{else}\{stmt_2\}$$

Formalizing programs - *Syntax*

$$x \in Vars$$
$$n \in \mathbb{N}$$
$$\ominus \in \{\neg, \ldots\}$$
$$\oplus \in \{+, -, *, \ldots\}$$

$$expr = n \mid x \mid \ominus\ expr \mid expr_1\ \oplus\ expr_2$$

$$stmt = \texttt{stop} \mid \texttt{skip} \mid x \leftarrow expr \mid stmt_1; stmt_2$$
$$\mid \texttt{if}(expr)\{stmt_1\}\texttt{else}\{stmt_2\}$$

Formalizing programs - *Syntax*

$$x \in Vars$$

$$n \in \mathbb{N}$$

$$\ominus \in \{\neg, \ldots\}$$

$$\oplus \in \{+, -, *, \ldots\}$$

$$expr = n \mid x \mid \ominus\ expr \mid expr_1\ \oplus\ expr_2$$

$$stmt = \texttt{stop} \mid \texttt{skip} \mid x \leftarrow expr \mid stmt_1; stmt_2$$

$$\mid \texttt{if}(expr)\{stmt_1\}\texttt{else}\{stmt_2\}$$

Formalizing programs - *Syntax*

$$x \in Vars$$

$$n \in \mathbb{N}$$

$$\ominus \in \{\neg, \ldots\}$$

$$\oplus \in \{+, -, *, \ldots\}$$

$$expr ::= n \mid x \mid \ominus\ expr \mid expr_1\ \oplus\ expr_2$$

$$stmt ::= \texttt{stop} \mid \texttt{skip} \mid x \leftarrow expr \mid stmt_1; stmt_2$$
$$\mid \texttt{if}(expr)\{stmt_1\}\texttt{else}\{stmt_2\}$$

Formalizing programs - *Semantics*

Formalizing programs - *Semantics*

- Program memories

$$\sigma \in Mems = Vars \to \mathbb{N}$$

Formalizing programs - *Semantics*

- Program memories

$$\sigma \in Mems = Vars \to \mathbb{N}$$

- Our goal: Programs transform memories

$$\langle\langle Prg \rangle\rangle : Mems \to Mems$$

Formalizing programs - *Semantics*

- Program memories

$$\sigma \in Mems = Vars \to \mathbb{N}$$

- Our goal: Programs transform memories

$$\langle\langle Prg \rangle\rangle : Mems \to Mems$$

- There are many ways of specifying program semantics
  - ***Small-step semantics***

Formalizing programs - ***Semantics***

- Program memories

$$\sigma \in Mems = Vars \to \mathbb{N}$$

- Our goal: Programs transform memories

$$\langle\langle Prg \rangle\rangle : Mems \to Mems$$

- There are many ways of specifying program semantics
  - ***Small-step semantics***

Define meaning of ***each instruction***! Derive program semantics ***compositionally***!

Small-step semantics - 1

Small-step semantics - 1

Evaluation of **`expressions`**: $[\![e]\!] : Mems \to \mathbb{N}$

Small-step semantics - 1

Evaluation of ***expressions***: $[\![e]\!] : Mems \to \mathbb{N}$

$$[\![n]\!](m) = n$$

$$[\![x]\!](m) = m(x)$$

$$[\![\ominus e]\!](m) = \ominus\, [\![e]\!](m)$$

$$[\![e_1 \oplus e_2]\!](m) = [\![e_1]\!](m) \oplus [\![e_2]\!](m)$$

Small-step semantics - 1

Evaluation of ***expressions***: $[\![e]\!] : Mems \to \mathbb{N}$

$$[\![n]\!](m) = n$$

$$[\![x]\!](m) = m(x)$$

$$[\![\ominus e]\!](m) = \ominus\, [\![e]\!](m)$$

$$[\![e_1 \oplus e_2]\!](m) = [\![e_1]\!](m) \oplus [\![e_2]\!](m)$$

Small-step semantics - 1

Evaluation of **`expressions`**: $[\![e]\!] : Mems \to \mathbb{N}$

$$[\![n]\!](m) = n$$

$$[\![x]\!](m) = m(x)$$

$$[\![\ominus e]\!](m) = \ominus\, [\![e]\!](m)$$

$$[\![e_1 \oplus e_2]\!](m) = [\![e_1]\!](m) \oplus [\![e_2]\!](m)$$

Small-step semantics - 1

Evaluation of ***expressions***: $[\![e]\!] : Mems \to \mathbb{N}$

$$[\![n]\!](m) = n$$

$$[\![x]\!](m) = m(x)$$

$$[\![\ominus e]\!](m) = \ominus\, [\![e]\!](m)$$

$$[\![e_1 \oplus e_2]\!](m) = [\![e_1]\!](m) \oplus [\![e_2]\!](m)$$

Small-step semantics - 2

Small-step semantics - 2

- The meaning of ***`each instruction`*** is formalized using one or more ***`inference rules`*** over program configurations

Small-step semantics - 2

- The meaning of ***each instruction*** is formalized using one or more ***inference rules*** over program configurations

- ***Program configurations***

$$\langle p, m \rangle \in Prg \times Mem$$

Small-step semantics - 2

- The meaning of **`each instruction`** is formalized using one or more **`inference rules`** over program configurations
- **`Program configurations`**

`Program` models the state of **`computation`**

$$\langle p, m \rangle \in Prg \times Mem$$

Small-step semantics - 2

- The meaning of **`each instruction`** is formalized using one or more **`inference rules`** over program configurations

- **`Program configurations`**

$$\langle p, m \rangle \in Prg \times Mem$$

`Program` models the state of **`computation`**

`Memory` models the state of the **`data`**

Small-step semantics - 2

- The meaning of ***each instruction*** is formalized using one or more ***inference rules*** over program configurations
- ***Program configurations***

$$\langle p, m \rangle \in Prg \times Mem$$

Program models the state of ***computation***

Memory models the state of the ***data***

- One application of the rules models a "***small step***" of the computation

Small-step semantics - 3

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

Small-step semantics - 2

$Conf = Prg \times Mem$

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

Small-step semantics - 2

$Conf = Prg \times Mem$

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

`Stop`

Small-step semantics - 2

$Conf = Prg \times Mem$

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

`Stop`

$$\frac{}{\langle \texttt{stop}, m \rangle \rightarrow \langle \texttt{stop}, m \rangle}$$

Small-step semantics - 2

$Conf = Prg \times Mem$

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

`Skip`

Small-step semantics - 2

$Conf = Prg \times Mem$

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

`Skip`

$$\frac{}{\langle \texttt{skip}, m \rangle \rightarrow \langle \texttt{stop}, m \rangle}$$

Small-step semantics - 2

$Conf = Prg \times Mem$

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

`Assignment`

Small-step semantics - 2

$Conf = Prg \times Mem$

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

`Assignment`

$$\frac{[\![e]\!](m) = v}{\langle x \leftarrow e, m\rangle \rightarrow \langle \texttt{stop}, m[x \mapsto v]\rangle}$$

Small-step semantics - 2

$Conf = Prg \times Mem$

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

`Sequential composition`

Small-step semantics - 2

$Conf = Prg \times Mem$

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

`Sequential composition`

$$\frac{\langle stmt_1, m\rangle \rightarrow \langle \texttt{stop}, m'\rangle}{\langle stmt_1; stmt_2, m\rangle \rightarrow \langle stmt_2, m'\rangle}$$

Small-step semantics - 2

$Conf = Prg \times Mem$

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

`Sequential composition`

$$\frac{\langle stmt_1, m\rangle \rightarrow \langle \texttt{stop}, m'\rangle}{\langle stmt_1; stmt_2, m\rangle \rightarrow \langle stmt_2, m'\rangle}$$

$$\frac{\langle stmt_1, m\rangle \rightarrow \langle stmt_1', m'\rangle \qquad stmt_1' \neq \texttt{stop}}{\langle stmt_1; stmt_2, m\rangle \rightarrow \langle stmt_1'; stmt_2, m'\rangle}$$

Small-step semantics - 2

$Conf = Prg \times Mem$

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

`If statements`

Small-step semantics - 2

$Conf = Prg \times Mem$

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

`If statements`

Small-step semantics - 2

$Conf = Prg \times Mem$

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

`If statements`

$$\frac{[\![e]\!] \neq 0}{\langle \texttt{if}(e)\{stmt_1\}\texttt{else}\{stmt_2\}, m\rangle \rightarrow \langle stmt_1, m\rangle}$$

Small-step semantics - 2

$Conf = Prg \times Mem$

Evaluation of **`statements`**: $\rightarrow \subseteq Conf \times Conf$

`If statements`

$$\frac{[\![e]\!] \neq 0}{\langle \texttt{if}(e)\{stmt_1\}\texttt{else}\{stmt_2\}, m\rangle \rightarrow \langle stmt_1, m\rangle}$$

$$\frac{[\![e]\!] = 0}{\langle \texttt{if}(e)\{stmt_1\}\texttt{else}\{stmt_2\}, m\rangle \rightarrow \langle stmt_2, m\rangle}$$

Small-step semantics - 4

Program semantics:

Small-step semantics - 4

Program semantics: $\langle\langle Prg \rangle\rangle : Mems \rightarrow Mems$

Small-step semantics - 4

Program semantics: $\langle\!\langle Prg \rangle\!\rangle : Mems \to Mems$

$$\langle\!\langle Prg \rangle\!\rangle(m) := \begin{cases} m' & \text{if } \langle Prg, m \rangle \to^* \langle \texttt{stop}, m' \rangle \\ \bot & \text{otherwise} \end{cases}$$

Small-step semantics - 4

Program semantics: $\langle\!\langle Prg \rangle\!\rangle : Mems \to Mems$

$$\langle\!\langle Prg \rangle\!\rangle(m) := \begin{cases} m' & if\ \langle Prg, m \rangle \to^* \langle \texttt{stop}, m' \rangle \\ \bot & otherwise \end{cases}$$

Partial function!

Small-step semantics - 4

Program semantics: $\langle\!\langle Prg \rangle\!\rangle : Mems \to Mems$

Transitive closure of $\to$

$$\langle\!\langle Prg \rangle\!\rangle(m) := \begin{cases} m' & \text{if } \langle Prg, m \rangle \to^* \langle \texttt{stop}, m' \rangle \\ \bot & \text{otherwise} \end{cases}$$

Partial function!

Programs have a precise meaning

😄

What is a secure program?

Attacker model

- Split variables into:
  - **`Public`** variables x_{public}
  - **`Secret`** variables x_{secret}

`Attacker` can **`observe`** final value of **`public`** variables

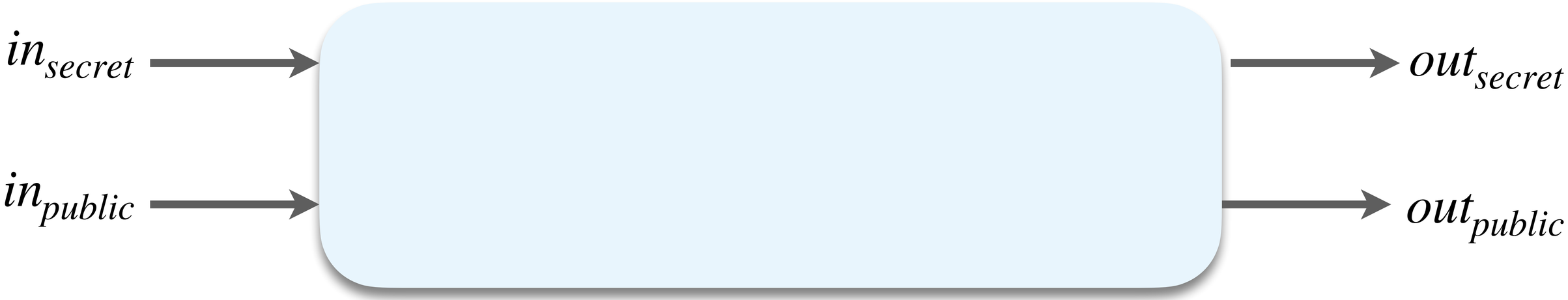
Non-interference (informally)

Non-interference (informally)

Security (confidentiality)

Attacker cannot derive secret values!

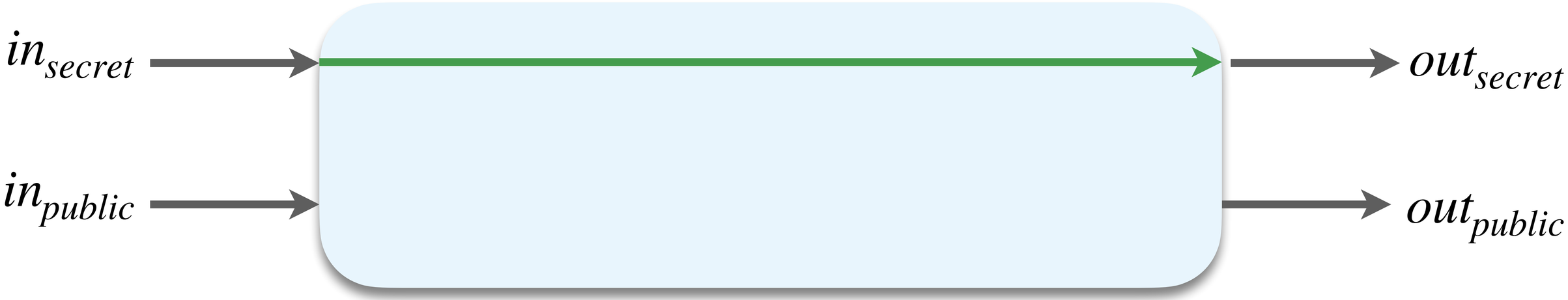
Non-interference (informally)



Security (confidentiality)

Attacker cannot derive secret values!

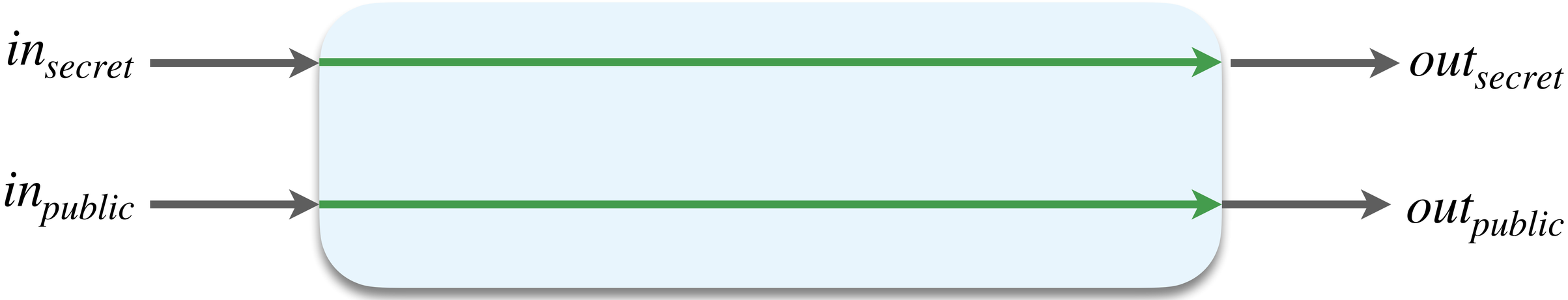
Non-interference (informally)



Security (confidentiality)

Attacker cannot derive secret values!

Non-interference (informally)



Security (confidentiality)

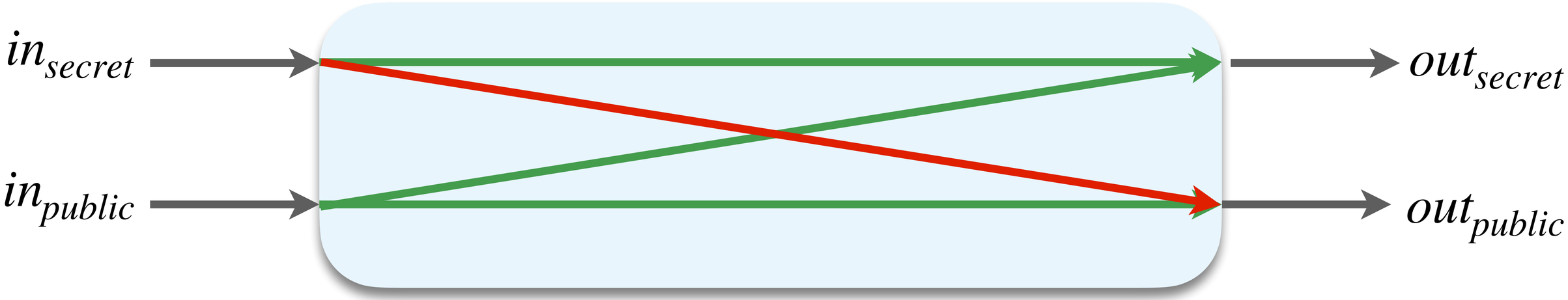
Attacker cannot derive secret values!

Non-interference (informally)

Security (confidentiality)

Attacker cannot derive secret values!

Non-interference (informally)



Security (confidentiality)

Attacker cannot derive secret values!

Non-interference (formally)

Non-interference (formally)

Two memories **`m`** and **`m'`** are **`public-equivalent`** $m \sim m'$ if

For all public variables $x_{public} \in Vars$

$$m(x_{public}) = m'(x_{public})$$

Non-interference (formally)

Two memories **`m`** and **`m'`** are ***`public-equivalent`*** $m \sim m'$ if

For all public variables $x_{public} \in Vars$

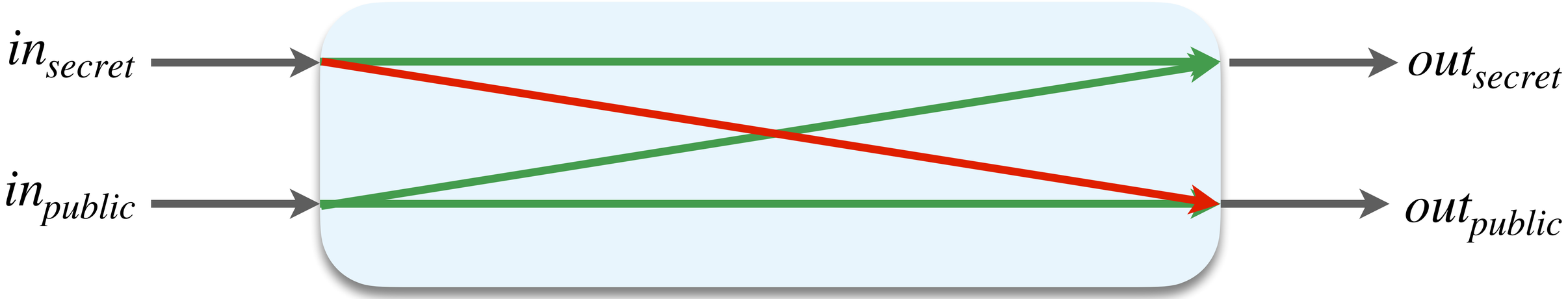
$$m(x_{public}) = m'(x_{public})$$

Program **`Prg`** is ***`non-interferent`*** if

For all memories **`m`** and **`m'`**:

$$m \sim m' \Rightarrow \langle\langle Prg \rangle\rangle(m) \sim \langle\langle Prg \rangle\rangle(m')$$

Non-interference (formally)



Program **`Prg`** is ***`non-interferent`*** if

For all memories **`m`** and **`m'`**:

$$m \sim m' \Rightarrow \langle\!\langle Prg \rangle\!\rangle(m) \sim \langle\!\langle Prg \rangle\!\rangle(m')$$

Non-interference — examples

```
program f1(xpublic, xsecret) {
  xpublic = xsecret + 1
}
```

Non-interference — examples

```
program f1(xpublic, xsecret) {
  xpublic = xsecret + 1
}
```

Non-interference — examples

```
program f2(xpublic, xsecret) {
  xsecret = 0
  xpublic = xsecret + 1
}
```

Non-interference — examples

```
program f2(xpublic, xsecret) {
  xsecret = 0
  xpublic = xsecret + 1
}
```

Non-interference — examples

```
program f3(xpublic, xsecret) {
  xsecret = xpublic + 1
}
```

Non-interference — examples

```
program f3(xpublic, xsecret) {
  xsecret = xpublic + 1
}
```

Non-interference — examples

```
program f4(xpublic, xsecret) {
  xsecret = 1
}
```

Non-interference — examples

```
program f4(xpublic, xsecret) {
  xsecret = 1
}
```

Non-interference — examples

```
program f5(x_public, x_secret) {
  if (x_secret==0)
    x_public = 1
  else
    x_public = 0
}
```

Non-interference — examples

```
program f5(xpublic, xsecret) {
  if (xsecret==0)
    xpublic = 1
  else
    xpublic = 0
}
```

We have a notion of security for programs

😄

What can we do with non-interference?

What can we do with NI?

- ***Non-interference*** provides theoretical foundations for talking about ***program security***

What can we do with NI?

- ***Non-interference*** provides theoretical foundations for talking about ***program security***

- We can ensure that ***programs are secure***:

What can we do with NI?

- ***Non-interference*** provides theoretical foundations for talking about ***program security***
- We can ensure that ***programs are secure***:
  - ***Static*** enforcement

What can we do with NI?

- ***Non-interference*** provides theoretical foundations for talking about ***program security***

- We can ensure that ***program***

  - ***Static*** enforcement

Check statically if programs are NI

What can we do with NI?

- ***Non-interference*** provides theoretical foundations for talking about ***program security***

- We can ensure that ***program***

  - ***Static*** enforcement

  - ***Dynamic*** enforcement

Check statically if programs are NI

What can we do with NI?

- ***Non-interference*** provides theoretical foundations for talking about ***program security***

- We can ensure that ***program***
  - ***Static*** enforcement
  - ***Dynamic*** enforcement

Check statically if programs are NI

Modify program ***behavior*** to enforce NI at ***runtime***

Static enforcement of NI

- Use ***program analysis*** techniques to check if programs are non-interferent:
  - Type systems
  - Symbolic execution
  - Abstract interpretation
  - …

Static enforcement of NI

- Use ***program analysis*** techniques to check if programs are non-interferent:
  - Type systems
  - Symbolic execution
  - Abstract interpretation
  - …

See lecture 5!

We can make programs secure

😄

Backup

A type system for NI

A type system for NI

- Context:

A type system for NI

- Context:

$$\Gamma(x_{\texttt{public}}) = \texttt{public}$$
$$\Gamma(x_{\texttt{secret}}) = \texttt{secret}$$

A type system for NI

- Context: $\Gamma(x_{\texttt{public}}) = \texttt{public}$ $\Gamma(x_{\texttt{secret}}) = \texttt{secret}$
- Typing rules:

Stop & Skip

$$\frac{}{pc \vdash \texttt{stop}} \qquad \frac{}{pc \vdash \texttt{skip}}$$

A type system for NI

- Context: $\Gamma(x_{\texttt{public}}) = \texttt{public}$
  $\Gamma(x_{\texttt{secret}}) = \texttt{secret}$
- Typing rules:

Stop & Skip

$$\frac{}{pc \vdash \texttt{stop}} \qquad \frac{}{pc \vdash \texttt{skip}}$$

$pc \in \{\texttt{public}, \texttt{secret}\}$

A type system for NI

- Context:

$$\Gamma(x_{\texttt{public}}) = \texttt{public}$$
$$\Gamma(x_{\texttt{secret}}) = \texttt{secret}$$

- Typing rules:

Assignment

$$\frac{\Gamma(e) \sqcup pc \sqsubseteq \Gamma(x)}{pc \vdash x \leftarrow e}$$

A type system for NI

- Context: $\Gamma(x_{\texttt{public}}) = \texttt{public}$ $\Gamma(x_{\texttt{secret}}) = \texttt{secret}$
- Typing rules:

Assignment

$$\frac{\Gamma(e) \sqcup pc \sqsubseteq \Gamma(x)}{pc \vdash x \leftarrow e}$$

$\texttt{public} \sqcup \texttt{public} = \texttt{public}$ $\texttt{public} \sqcup \texttt{secret} = \texttt{secret}$

$\texttt{secret} \sqcup \texttt{public} = \texttt{secret}$ $\texttt{secret} \sqcup \texttt{secret} = \texttt{secret}$

A type system for NI

- Context: $\Gamma(x_{\texttt{public}}) = \texttt{public}$, $\Gamma(x_{\texttt{secret}}) = \texttt{secret}$
- Typing rules:

$\texttt{public} \sqsubseteq \texttt{secret}$
$\texttt{secret} \not\sqsubseteq \texttt{public}$

Assignment

$$\frac{\Gamma(e) \sqcup pc \sqsubseteq \Gamma(x)}{pc \vdash x \leftarrow e}$$

$\texttt{public} \sqcup \texttt{public} = \texttt{public}$
$\texttt{secret} \sqcup \texttt{public} = \texttt{secret}$
$\texttt{public} \sqcup \texttt{secret} = \texttt{secret}$
$\texttt{secret} \sqcup \texttt{secret} = \texttt{secret}$

A type system for NI

- Context:

$$\Gamma(x_{\texttt{public}}) = \texttt{public}$$
$$\Gamma(x_{\texttt{secret}}) = \texttt{secret}$$

- Typing rules:

Sequential composition

$$\frac{pc \vdash stmt_1 \qquad pc \vdash stmt_2}{pc \vdash stmt_1; stmt_2}$$

A type system for NI

- Context:

$$\Gamma(x_{\texttt{public}}) = \texttt{public}$$
$$\Gamma(x_{\texttt{secret}}) = \texttt{secret}$$

- Typing rules:

If statements

$$\frac{pc \sqcup \Gamma(e) \vdash stmt_1 \qquad pc \sqcup \Gamma(e) \vdash stmt_2}{pc \vdash \texttt{if } (e) \ \{stmt_1\} \ \texttt{else } \{stmt_2\}}$$

A type system for NI

- Context: $\Gamma(x_{\texttt{public}}) = \texttt{public}$
  $\Gamma(x_{\texttt{secret}}) = \texttt{secret}$
- Typing rules:

For all ***programs*** Prg, if $\texttt{public} \vdash Prg$ then Prg satisfies non-interference

Dynamic enforcement of NI

Dynamic enforcement of NI

- ***`Modify`*** program ***`semantics`*** to enforce non-interference at runtime

Dynamic enforcement of NI

- ***Modify*** program ***semantics*** to enforce non-interference at runtime

- Context $\Delta : \mathit{Vars} \cup \{pc\} \to \{\texttt{public}, \texttt{secret}\}$

Assignment

$$\frac{[\![e]\!](m) = v \qquad \Delta(pc) \sqcup \Delta(e) \sqsubseteq \Delta(x)}{\langle x \leftarrow e, m, \Delta \rangle \Rightarrow \langle \texttt{stop}, m[x \mapsto v], \Delta[x \mapsto \Delta(e) \sqcup \Delta(pc)] \rangle}$$

Dynamic enforcement of NI

- ***Modify*** program ***semantics*** to enforce non-interference at runtime

- Context $\Delta : Vars \cup \{pc\} \rightarrow \{\texttt{public}, \texttt{secret}\}$

If statements

$$\frac{[\![e]\!] \neq 0}{\langle \texttt{if}(e)\{stmt_1\}\texttt{else}\{stmt_2\}, m, \Delta\rangle \Rightarrow \langle stmt_1, m, \Delta[pc \mapsto \Delta(pc) \sqcup \Delta(e)]\rangle}$$

Dynamic enforcement of NI

- ***Modify*** program ***semantics*** to enforce non-interference at runtime

- Context $\Delta : Vars \cup \{pc\} \rightarrow \{\texttt{public}, \texttt{secret}\}$

For all ***programs*** Prg, executing Prg under the semantics $\Rightarrow$ satisfies non-interference

$$\langle \texttt{if}(e)\{stmt_1\}\texttt{else}\{stmt_2\}, m, \Delta\rangle \Rightarrow \langle stmt_1, m, \Delta[pc \mapsto \Delta(pc) \sqcup \Delta(e)]\rangle$$