Asymmetric Secure Multi-Execution with Declassification

Iulia Bolosteanu, Deepak Garg

Eindhoven, 4th of April 2016

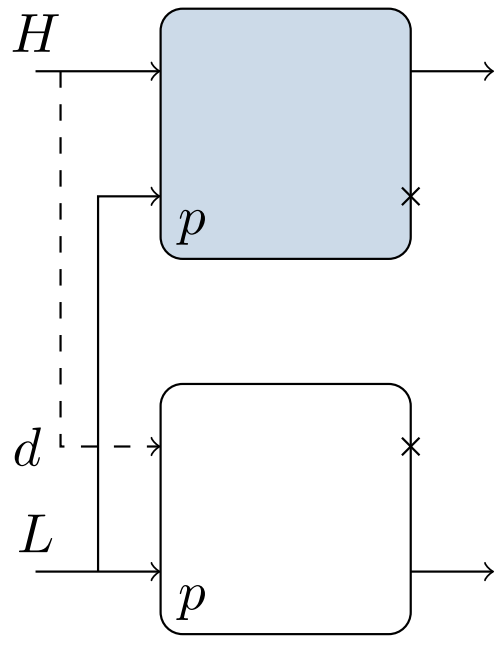
Max
Planck
Institute
for
Software Systems

Non-interference

- non-interference: standard information-flow policy
  - low outputs not influenced by high inputs
- enforcement: static, dynamic or hybrid analyses of programs
  - downside: not sound and precise simultaneously
- secure multi-execution: both sound and precise [DP10]
  - tradeoff: more computational power

Secure Multi-Execution (SME)

- two runs of the program: high and low
- high run: low and high inputs; only high outputs
- low run: only low inputs; only low outputs
- default inputs (0 or *null*) instead of high inputs



- non-interference by construction
- if p non-interferent, then SME precise

Secure Multi-Execution

Example 1

$\text{input}_H(x);$
$\text{input}_L(y);$
$a := x + 2;$
$b := y + a;$
$\text{output}_H(a);$
$\text{output}_L(b);$

flow from high input x to low output b!

Secure Multi-Execution

Example 1

$\text{input}_H(x);$	5
$\text{input}_L(y);$	3
$a := x + 2;$	$5 + 2$
$b := y + a;$	$3 + 7$
$\text{output}_H(a);$	7
$\text{output}_L(b);$	10

flow from high input x to low output b!

Secure Multi-Execution

Example 1

		High SME run:		**Low SME run:**	
$\text{input}_H(x);$	5	$\text{input}_H(5);$		$\text{input}_H(0);$	
$\text{input}_L(y);$	3	$\text{input}_L(3);$		$\text{input}_L(3);$	
$a := x + 2;$	$5 + 2$	$a := x + 2;$	$5 + 2$	$a := x + 2;$	$0 + 2$
$b := y + a;$	$3 + 7$	$b := y + a;$	$7 + 3$	$b := y + a;$	$3 + 2$
$\text{output}_H(a);$	7	$\text{output}_H(7);$		~~$\text{output}_H(2);$~~	
$\text{output}_L(b);$	10	~~$\text{output}_L(10);$~~		$\text{output}_L(5);$	

flow from high input x to low output b!

- SME enforces non-interference:
  - irrespective of the value of input x, output b is always 5

- precision not achieved, as program insecure

Secure Multi-Execution

Example 2

$\text{input}_H(x);$
$\text{input}_L(y);$
$a := x + 2;$
$b := y + 3;$
$\text{output}_H(a);$
$\text{output}_L(b);$

Secure Multi-Execution

Example 2

		High SME run:		**Low SME run:**	
$\text{input}_H(x);$	5	$\text{input}_H(5);$		$\text{input}_H(0);$	
$\text{input}_L(y);$	3	$\text{input}_L(3);$		$\text{input}_L(3);$	
$a := x + 2;$	$5 + 2$	$a := x + 2;$	$5 + 2$	$a := x + 2;$	$0 + 3$
$b := y + 3;$	$3 + 3$	$b := y + 3;$	$3 + 3$	$b := y + 3;$	$3 + 3$
$\text{output}_H(a);$	7	$\text{output}_H(7);$		~~$\text{output}_H(7);$~~	
$\text{output}_L(b);$	6	~~$\text{output}_L(6);$~~		$\text{output}_L(6);$	

- program non-interference implies SME precision

Secure Multi-Execution

Drawbacks

		High SME run:	**Low SME run:**
$\text{input}_H(x);$	5	$\text{input}_H(5);$	$\text{input}_H(0);$
$\text{input}_L(y);$	3	$\text{input}_L(3);$	$\text{input}_L(3);$
$a := x + 2;$		$a := x + 2;$	$a := x + 2;$
$b := y/x;$		$b := y/x;$	$b := y/x;$ ↯
$\text{output}_H(a);$	7	$\text{output}_H(7);$	~~$\text{output}_H(7);$~~
$\text{output}_L(b);$	3/5	~~$\text{output}_L(3/5);$~~	$\text{output}_L(???);$

- default values 0 or *null* not always a fortunate choice

Secure Multi-Execution

Drawbacks

		High SME run:	**Low SME run:**
$\text{input}_H(x);$	5	$\text{input}_H(5);$	$\text{input}_H(0);$
$\text{input}_L(y);$	3	$\text{input}_L(3);$	$\text{input}_L(3);$
$a := x + 2;$		$a := x + 2;$	$a := x + 2;$
$b := y/x;$		$b := y/x;$	$b := y/x;$ ↯
$\text{output}_H(a);$	7	$\text{output}_H(7);$	~~$\text{output}_H(7);$~~
$\text{output}_L(b);$	3/5	~~$\text{output}_L(3/5);$~~	$\text{output}_L(???);$

- default values 0 or *null* not always a fortunate choice
- execution under SME may lead to program crashing
- SME cannot be successfully applied to all programs
- implicit assumption: program robust to change in input

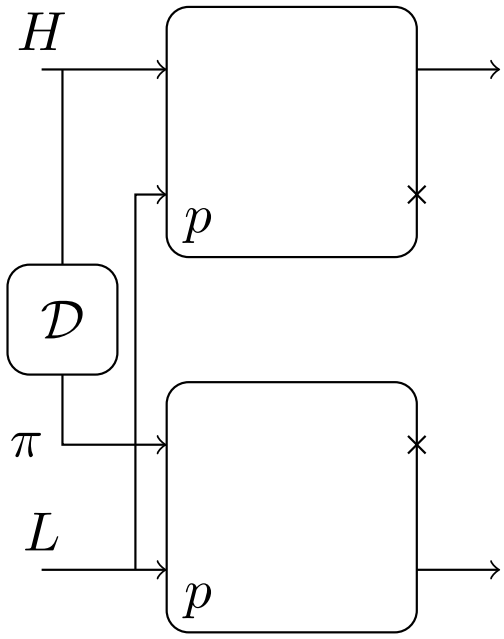
Secure Multi-Execution

- non-interference: low outputs not influenced by high inputs
  - too strict for practical applications
- intended release of some sensitive information is desirable
  - declassification needed
- combination between SME and declassification promising
  - non-reactive setting [RS13]
  - reactive setting [VGDPR14]

Secure Multi-Execution with Declassification [VGDPR14]

Reactive setting

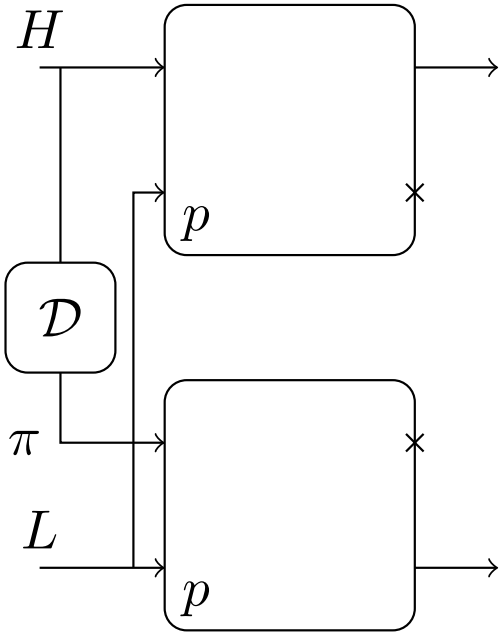
- stateful declassification policy ($\mathcal{D}$) external to application
- low run: declassified values instead of high inputs/default values



Secure Multi-Execution with Declassification [VGDPR14]

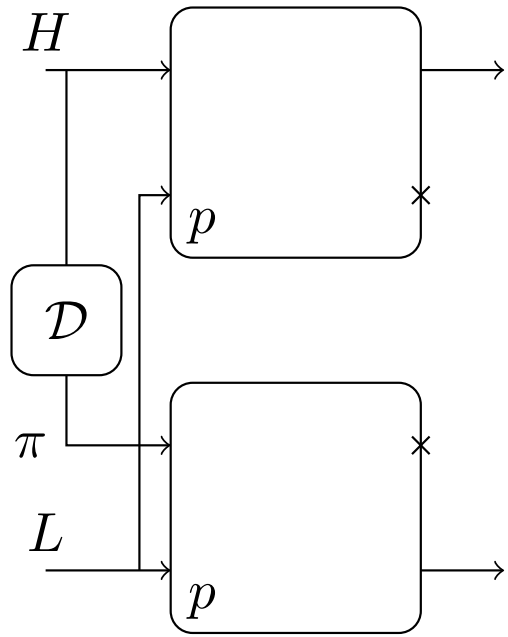
Reactive setting

- stateful declassification policy ($\mathcal{D}$) external to application
- low run: declassified values instead of high inputs/default values
- declassified values and high inputs may have different types
- program may crash or produce unexpected results



Secure Multi-Execution with Declassification [VGDPR14]

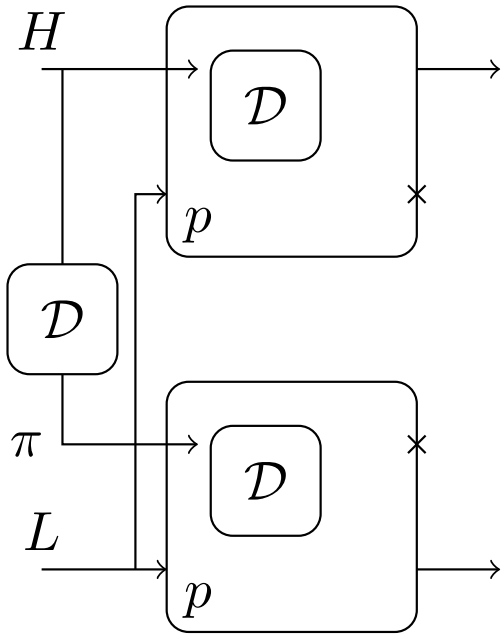
Reactive setting – Drawbacks



Secure Multi-Execution with Declassification [VGDPR14]

Reactive setting – Drawbacks

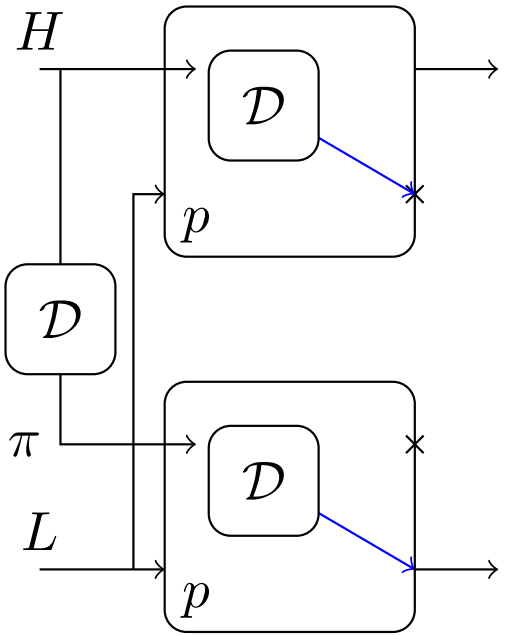
- correct program: contains semantically same declassification policy



Secure Multi-Execution with Declassification [VGDPR14]

Reactive setting – Drawbacks

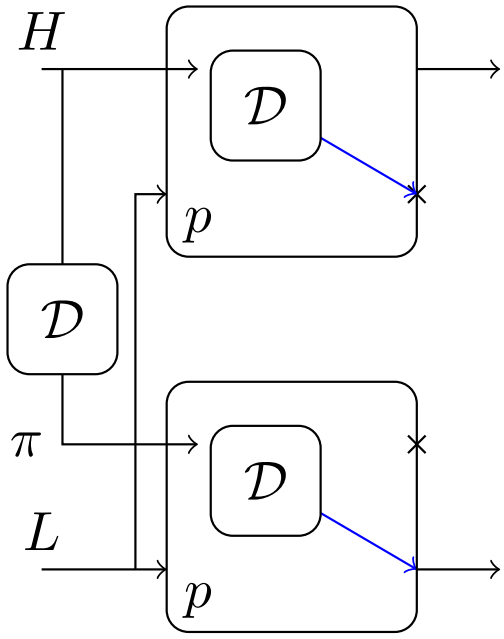
- correct program: contains semantically same declassification policy
- semantically, $\mathcal{D}$ lies on the path from the high inputs to low outputs



Secure Multi-Execution with Declassification [VGDPR14]

Reactive setting – Drawbacks

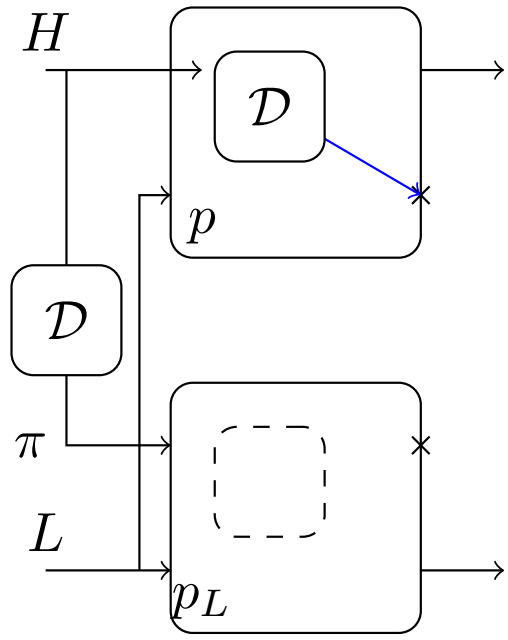
- correct program: contains semantically same declassification policy
- semantically, $\mathcal{D}$ lies on the path from the high inputs to low outputs
- precision for low outputs: only declassification is idempotent ($\pi(\pi(x)) = \pi(x)$)
- many declassification policies do not satisfy idempotence:
  - e.g.: hashing, encryption, adding noise



Recap

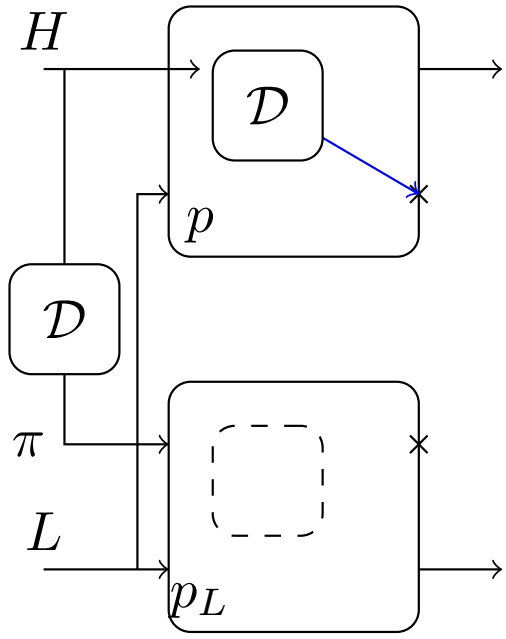
- SME promising dynamic technique for enforcing non-interference
- runs two copies of *same* program: high run and low run
- *different* inputs: original inputs to high run, modified inputs to low run
- implicit assumption: program robust to any change in input
- SME with declassification executes policy twice in low run (idempotence required)

Asymmetric-SME with Declassification



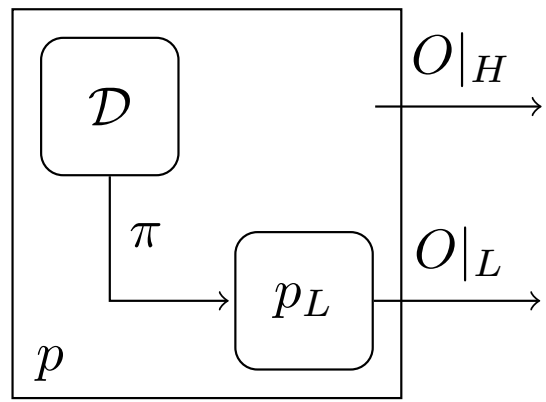
Asymmetric-SME with Declassification

- **different** programs in the two runs:
  - high run: original program; original inputs
  - low run: low slice p_L; declassified inputs
- p_L modified version of p crafted to respond correctly to declassified data



Low slice – Intuition

- if p compliant with $\mathcal{D}$, then low outputs dependent only on declassified data
- in addition p can be mathematically factorized in policy $\mathcal{D}$ and low slice p_L
- low slice handles only declassified data and produces only low outputs



Low slice – Example

$$
\begin{aligned}
&\text{input}_H(IP);\\
&y = \text{first8}(IP);\\
&z = \text{determine\_region}(y);\\
&\text{output}_L(z);
\end{aligned}
$$

Low slice – Example

$$\begin{array}{l} \text{input}_H(IP); \\ y = \text{first8}(IP); \\ z = \text{determine\_region}(y); \\ \text{output}_L(z); \end{array}$$

$$\mathcal{D}: \quad \begin{array}{l} \text{input}_H(IP); \\ \text{output}_H(\text{first8}(IP)); \end{array}$$

Low slice – Example

$$\begin{array}{l} \text{input}_H(IP); \\ y = \text{first8}(IP); \\ z = \text{determine\_region}(y); \\ \text{output}_L(z); \end{array}$$

$$\mathcal{D}: \quad \begin{array}{l} \text{input}_H(IP); \\ \text{output}_H(\text{first8}(IP)); \end{array} \qquad p_L: \quad \begin{array}{l} \text{input}_H(y); \\ z = \text{determine\_region}(y); \\ \text{output}_L(z); \end{array}$$

Low slice – Example

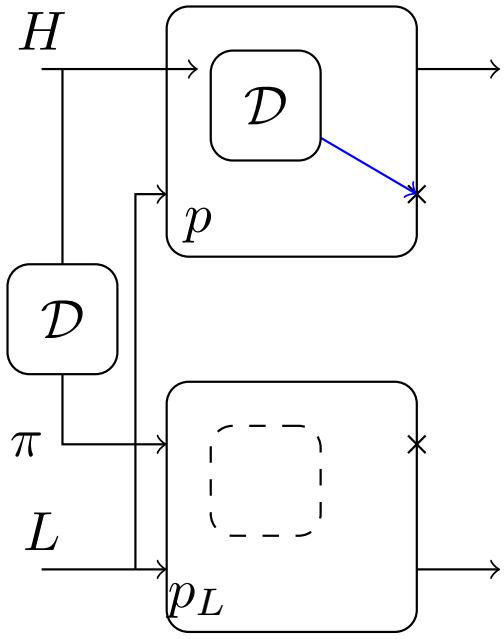
$$\begin{array}{l}\text{input}_H(IP);\\ y = \text{first8}(IP);\\ z = \text{determine\_region}(y);\\ \text{output}_L(z);\end{array}$$

$$\mathcal{D}: \begin{array}{l}\text{input}_H(IP);\\ \text{output}_H(\text{first8}(IP));\end{array} \qquad p_L: \begin{array}{l}\text{input}_H(y);\\ z = \text{determine\_region}(y);\\ \text{output}_L(z);\end{array}$$

- not obvious for all programs how to extract low slice;
- programmer intervention needed

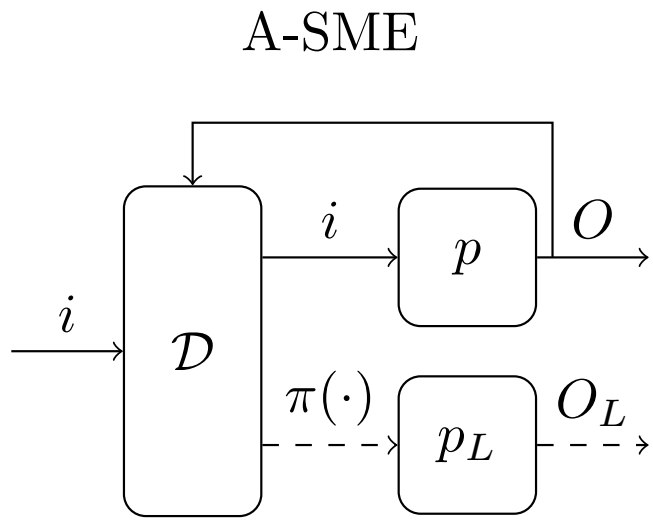
Asymmetric-SME with Declassification

- what kind of declassification policies can we enforce?
- can we prove A-SME security?
- can we prove A-SME precision?
- does the low slice always exist?



(Stateful) Declassification policies

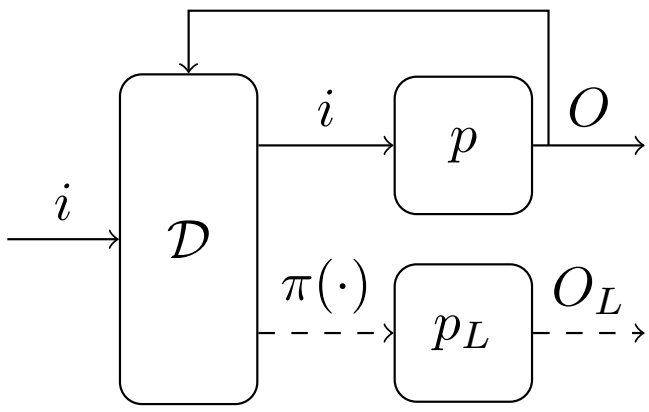
- A-SME enforces arbitrary policies with internal state



(Stateful) Declassification policies

- A-SME enforces arbitrary policies with internal state
- aggregate information about past inputs
  — also in [VGDPR14]
  e.g.: average of every 10 inputs

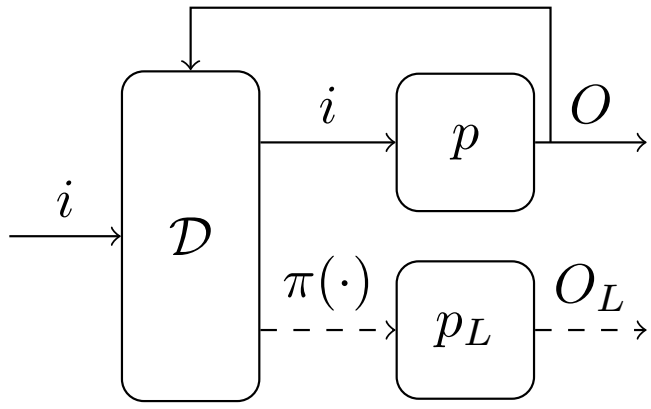
A-SME



(Stateful) Declassification policies

- A-SME enforces arbitrary policies with internal state
- aggregate information about past inputs — also in [VGDPR14]
  e.g.: average of every 10 inputs
- state-dependent input presence NEW
  e.g.: first 9 inputs high presence, 10th input low presence

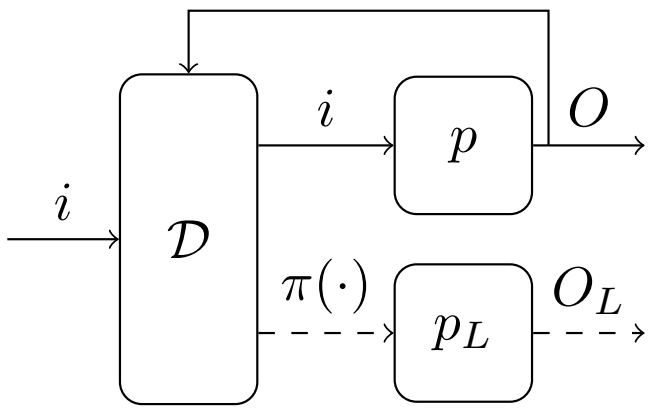
A-SME



(Stateful) Declassification policies

- A-SME enforces arbitrary policies with internal state
- aggregate information about past inputs
  — also in [VGDPR14]
  e.g.: average of every 10 inputs
- state-dependent input presence NEW
  e.g.: first 9 inputs high presence, 10th input low presence
- output feedback NEW
  e.g.: audit process reads non-identifying information from 100 client records and singles out one record to be fully declassified

A-SME



Security of A-SME

Theorem (Security, non-interference under $\mathcal{D}$)

Suppose $I_1, \mu_1 \longrightarrow_p E_1$ *and* $I_2, \mu_2 \longrightarrow_p E_2$ *and* $\mathcal{D}^*(s_1, E_1) = \mathcal{D}^*(s_2, E_2)$. *If* $I_1, s_1, \mu_1, \mu_L \Longmapsto^{\mathcal{D}}_{p,\, p_L} E_1'$ *and* $I_2, s_2, \mu_2, \mu_L \Longmapsto^{\mathcal{D}}_{p,\, p_L} E_2'$, *then* $E_1'|_o|_L = E_2'|_o|_L$.

- given two standard executions of a program leading to same declassified values, the A-SME execution of the program will produce same low outputs.

- i.e. irrespective of the correctness of the low slice, A-SME is always secure under any $\mathcal{D}$

Precision of A-SME

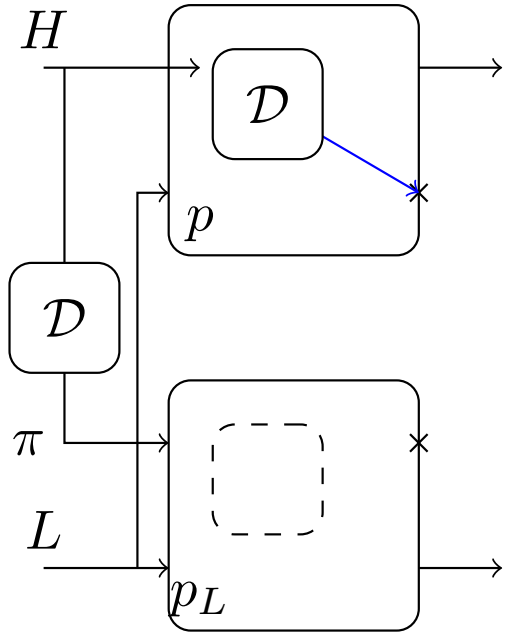
Theorem (Precision)

For any programs p and p_L, declassification policy $\mathcal{D}$ with initial state s and input list I, if $I, \mu_H \longrightarrow_p E$ and $I, s, \mu_H, \mu_L \Longmapsto^{\mathcal{D}}_{p,\, p_L} E'$, and (μ_L, p_L) is a correct low pair with respect to $\mathcal{D}$, s, p and μ_H, then $E|_o|_L = E'|_o|_L$ and $E|_o|_H = E'|_o|_H$.

- precision for high outputs: trivial
- high run of A-SME mimics standard execution

Precision for low outputs

- p_L produces low outputs of A-SME
- correct p_L: produces low outputs of p given declassified inputs from $\mathcal{D}$
- precision for low outputs: if p_L is correct



Definition (Correct low slice/correct low pair)

A program p_L of type Declassified × Memory ↦ [Output] × Memory and an initial memory μ_L are called a correct low pair (and p_L is called a correct low slice) with respect to policy $\mathcal{D}$, initial state s, program p and initial memory μ if for all inputs I, if $I, \mu \longrightarrow_p E$ and $\mathcal{D}^*(s, E) = R$ and $R, \mu_L \longrightarrow_{p_L} E'$, then $E|_o|_L = E'|_o|_L$.

Existence of correct low slice

Theorem (Existence of correct low slice)

If program p, starting from initial memory μ, does not leak outside declassification in policy $\mathcal{D}$ and initial state s, then there exists p_L and μ_L such that (μ_L, p_L) is a correct low pair with respect to $\mathcal{D}$, s, p, and μ.

- a correct low slice always exists if p conforms to policy $\mathcal{D}$
- i.e. two standard program executions which have same declassified values produce same low outputs
- proof does not give an algorithm for constructing the low slice
  - automated slicing feasible in many cases

Conclusion

- SME promising approach for enforcing non-interference
- introduced Asymmetric-SME: original program in high run, low slice in low run
- increased expressivity of declassification policies: output feedback and state-dependent input presence
- proved A-SME is always secure and precise only given a correct low slice
- showed a correct low slice always exists (in theory) if the program conforms to the policy

Thank you!

- SME promising approach for enforcing non-interference
- introduced Asymmetric-SME: original program in high run, low slice in low run
- increased expressivity of declassification policies: output feedback and state-dependent input presence
- proved A-SME is always secure and precise only given a correct low slice
- showed a correct low slice always exists (in theory) if the program conforms to the policy