Enforcing Opacity in Modular Systems

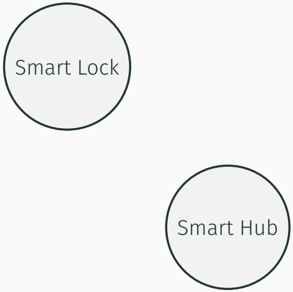
Graeme Zinck, Laurie Ricker, Hervé Marchand, Loïc Hélouët

2 July 2019

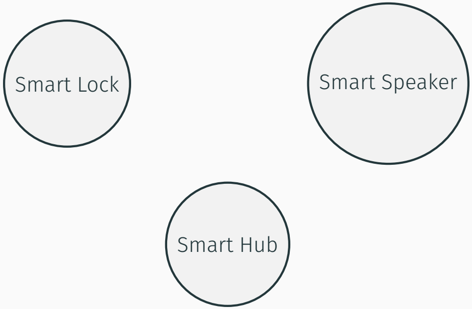
Mount Allison University, New Brunswick, Canada
Inria Rennes-Bretagne Atlantique, Rennes, France

Motivation

Motivation



Motivation



Motivation

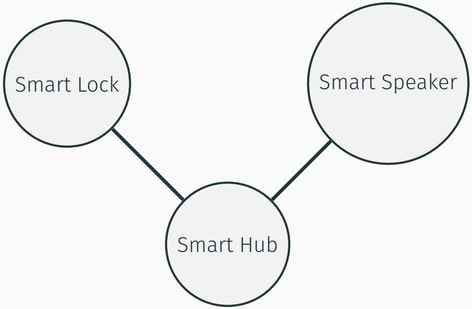


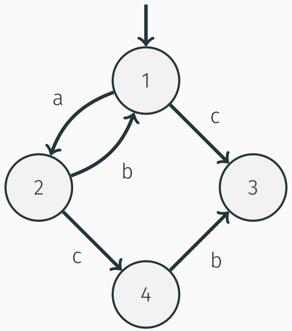
Table of contents

Model

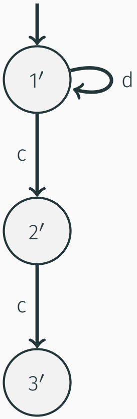
Model

1. System has multiple **modules** (automata)
2. Each module has its a set of **secret states**
3. Modules interact with **shared events**

Model

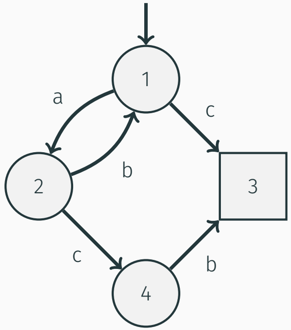


(a) G_1

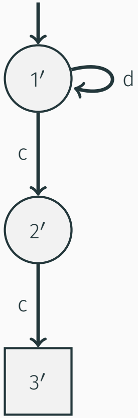


(b) G_2

Model



(a) G_1



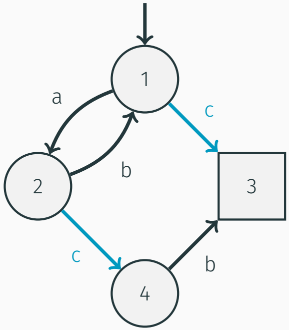
(b) G_2

Model

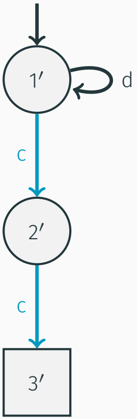
We construct the global system with the **parallel composition**

1. Synchronize on common events
2. Private events are interleaved

Model



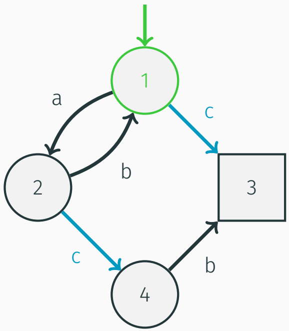
(a) G_1



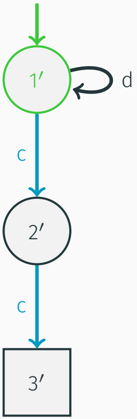
(b) G_2

(c) $G_1||G_2$

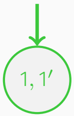
Model



(a) G_1

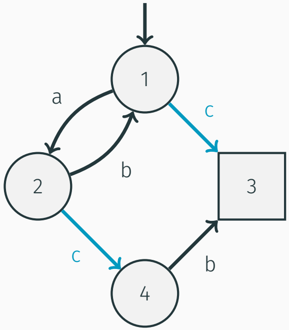


(b) G_2

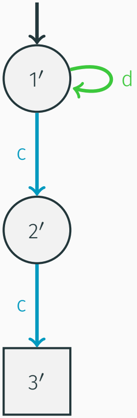


(c) $G_1||G_2$

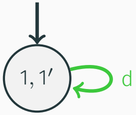
Model



(a) G_1

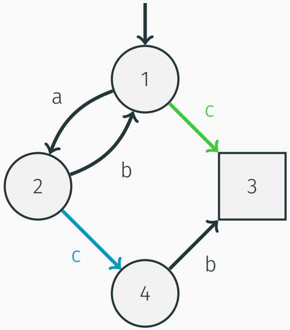


(b) G_2

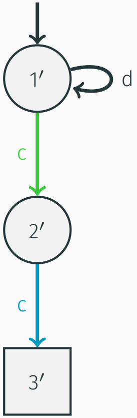


(c) $G_1||G_2$

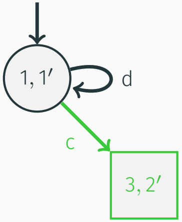
Model



(a) G_1

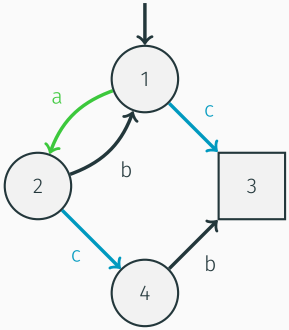


(b) G_2

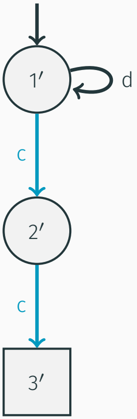


(c) $G_1||G_2$

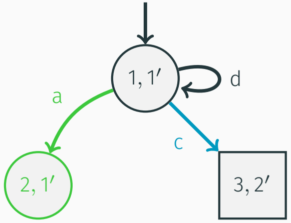
Model



(a) G_1

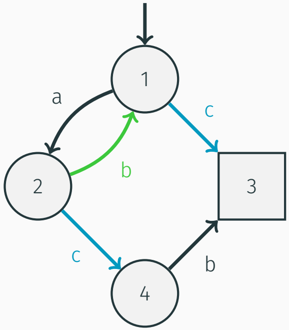


(b) G_2

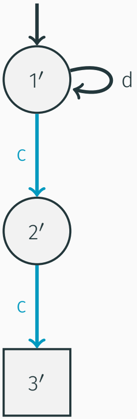


(c) $G_1||G_2$

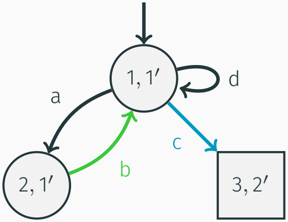
Model



(a) G_1

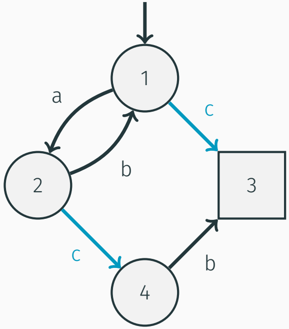


(b) G_2

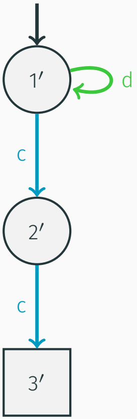


(c) $G_1||G_2$

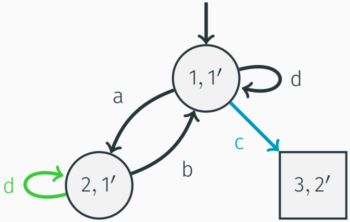
Model



(a) G_1

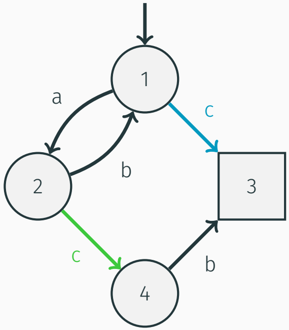


(b) G_2

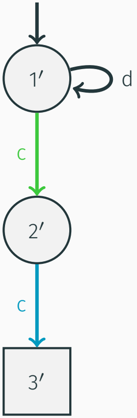


(c) $G_1||G_2$

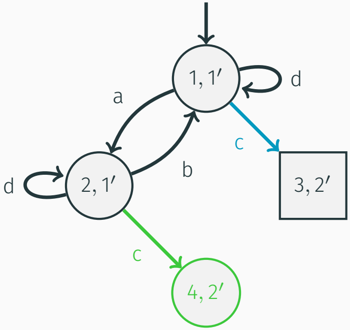
Model



(a) G_1

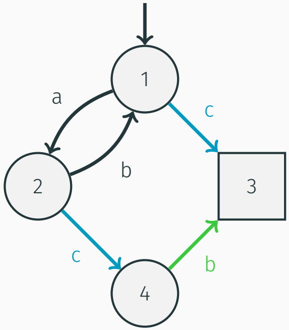


(b) G_2

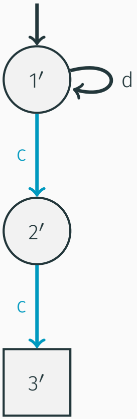


(c) $G_1||G_2$

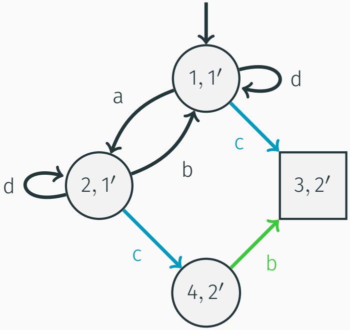
Model



(a) G_1



(b) G_2

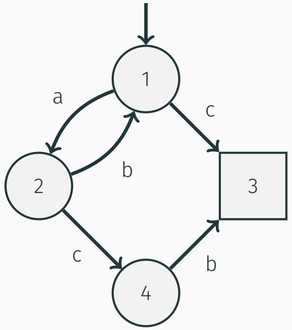


(c) $G_1||G_2$

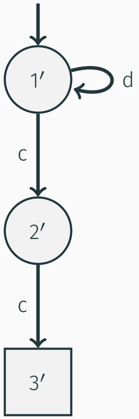
Attacker

- Modules are under **partial observation**
- Full knowledge of system architecture
- Has an **observable alphabet**

Model

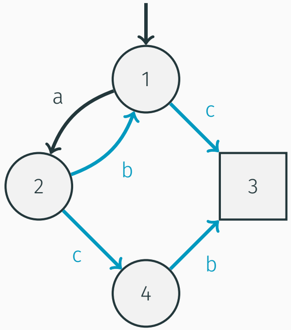


(a) G_1

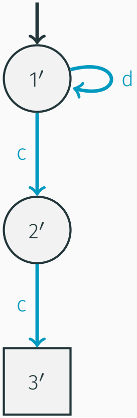


(b) G_2

Model



(a) G_1



(b) G_2

Opacity

Opacity

- **Opacity** models a form of security called **information flow**

Opacity

- **Opacity** models a form of security called **information flow**
- Attacker should not be able to distinguish a **secret state** from a **non-secret state**

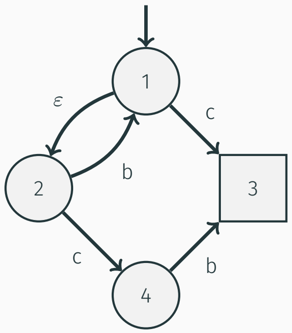
Opacity

- **Opacity** models a form of security called **information flow**
- Attacker should not be able to distinguish a **secret state** from a **non-secret state**
- *How can we verify that the attacker cannot distinguish secret states?*

Opacity

- **Opacity** models a form of security called **information flow**
- Attacker should not be able to distinguish a **secret state** from a **non-secret state**
- *How can we verify that the attacker cannot distinguish secret states?*
- **Determinization**: we treat events not in the attacker’s alphabet as epsilon events and determinize the system

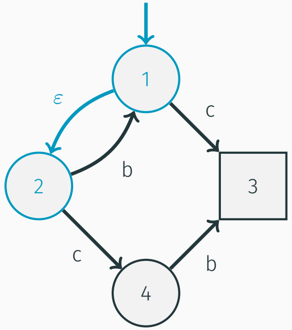
Determinizing G_1



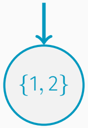
(a) G_1

(b) $Det_{\Sigma_a}(G_1)$

Determinizing G_1

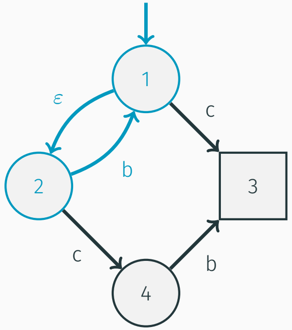


(a) G_1

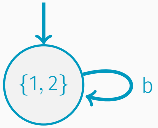


(b) $Det_{\Sigma_a}(G_1)$

Determinizing G_1

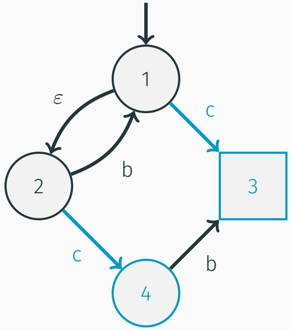


(a) G_1

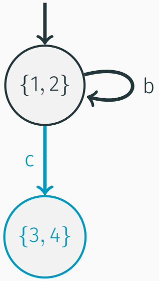


(b) $Det_{\Sigma_a}(G_1)$

Determinizing G_1

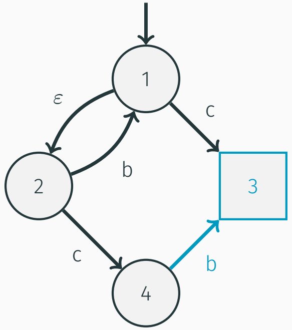


(a) G_1

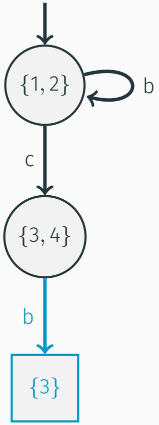


(b) $Det_{\Sigma_a}(G_1)$

Determinizing G_1

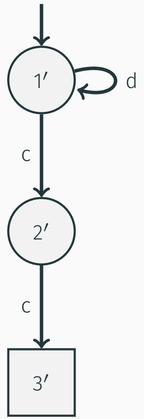


(a) G_1

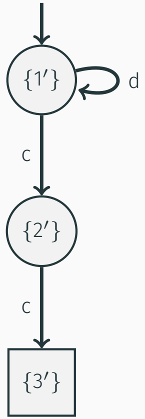


(b) $Det_{\Sigma_a}(G_1)$

Determinizing G_2



(a) G_2



(b) $Det_{\Sigma_a}(G_2)$

Opacity

- A secret is **opaque** if there are no states that are a subset of the set of secret states in the determinized automaton.

Opacity

- A secret is **opaque** if there are no states that are a subset of the set of secret states in the determinized automaton.
- This implies a simple PSPACE-complete verification algorithm

Opacity

Traditionally, we verify opacity in the global system as follows:

Opacity

Traditionally, we verify opacity in the global system as follows:

- Compose all modules

Opacity

Traditionally, we verify opacity in the global system as follows:

- Compose all modules
- Determinize the result

Opacity

Traditionally, we verify opacity in the global system as follows:

- Compose all modules
- Determinize the result
- Check if there are states that are a subset of the set of secret states

Opacity

What if we could check opacity locally, allowing us to avoid constructing the full system?

Verifying Opacity in Modular Systems

Verifying Opacity

- Can we verify opacity in a local module instead of verifying it globally?

Verifying Opacity

- Can we verify opacity in a local module instead of verifying it globally?
- Not necessarily...

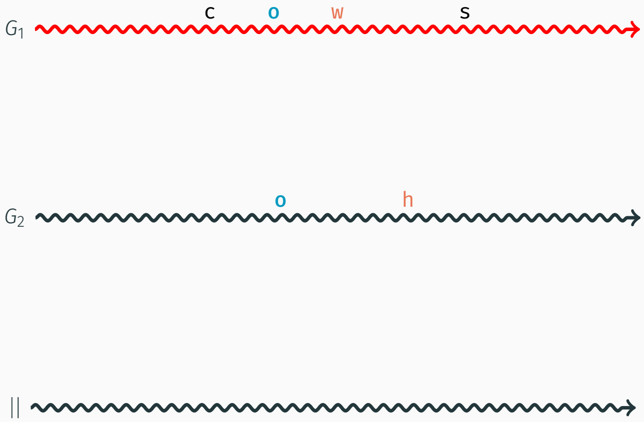
Verifying Opacity

- Can we verify opacity in a local module instead of verifying it globally?
- **Not necessarily...** but there is a sufficient condition which enables us to verify locally

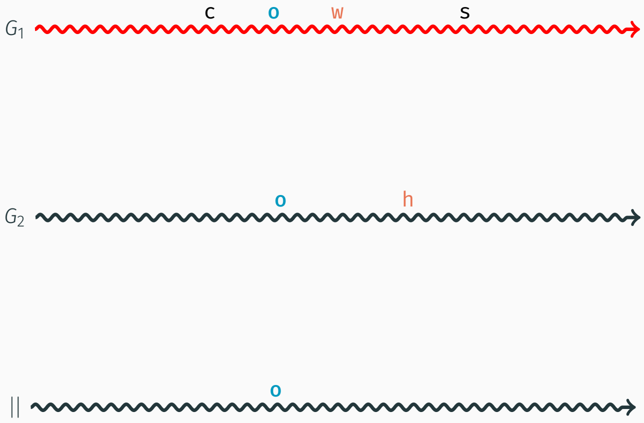
Verifying Opacity

- Can we verify opacity in a local module instead of verifying it globally?
- **Not necessarily...** but there is a sufficient condition which enables us to verify locally
- We can assume the **attacker observes the interface** between modules
- Assuming this allows us to check opacity of the local module's secret
    - If opaque locally, it's opaque for the composed system
    - Otherwise, we need to check the global system

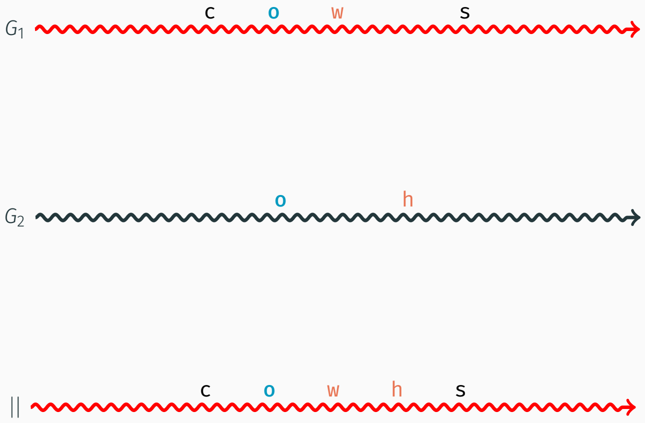
Verifying Opacity



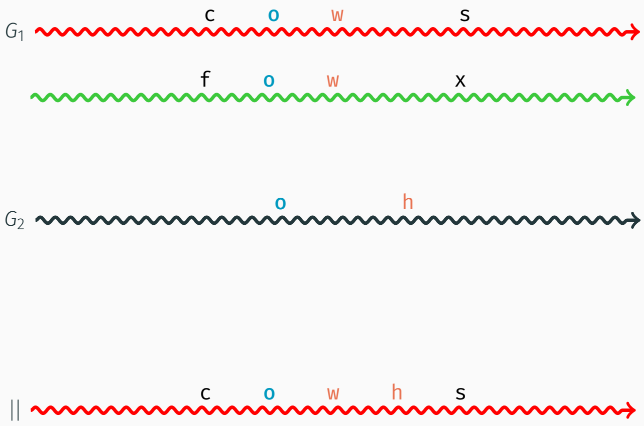
Verifying Opacity



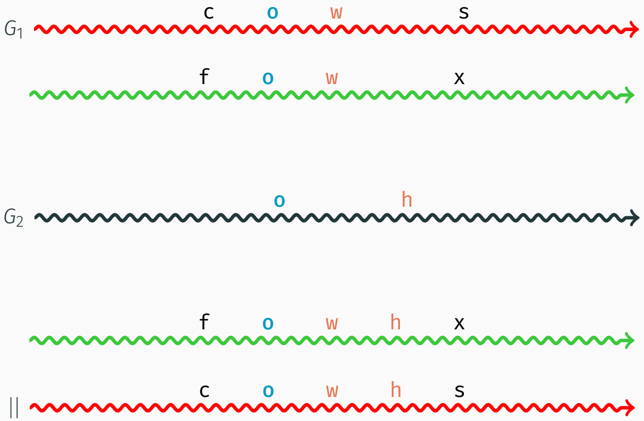
Verifying Opacity



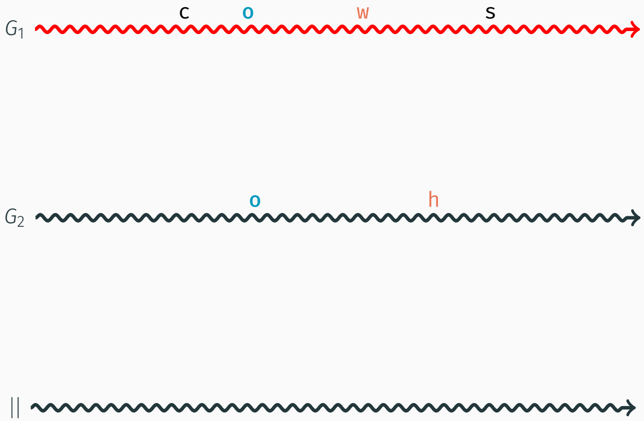
Verifying Opacity



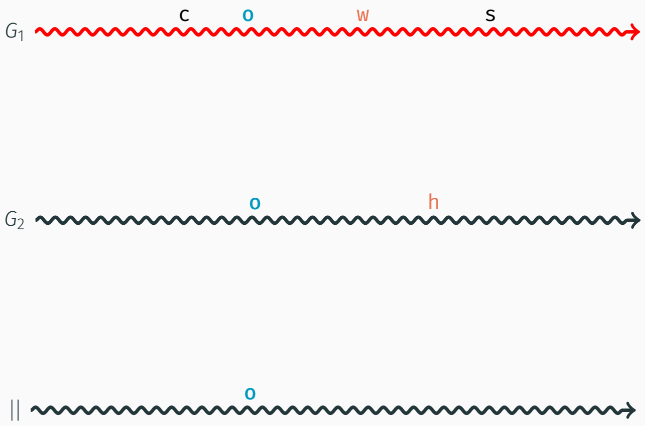
Verifying Opacity



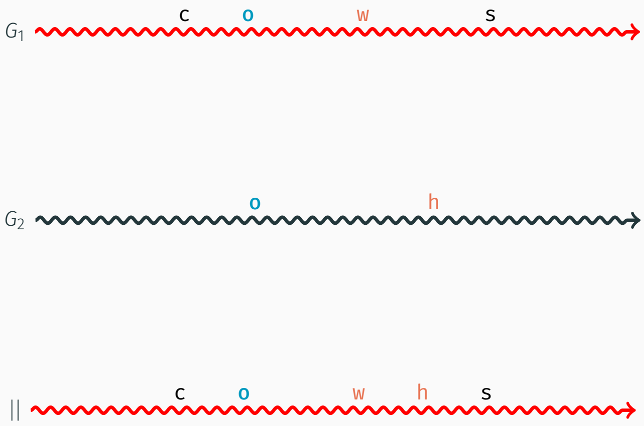
Verifying Opacity



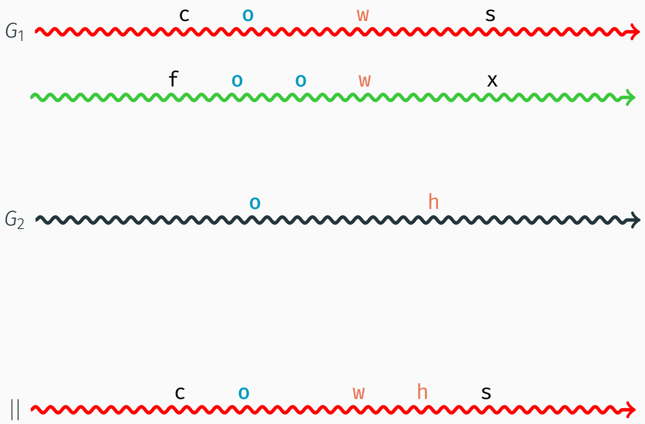
Verifying Opacity



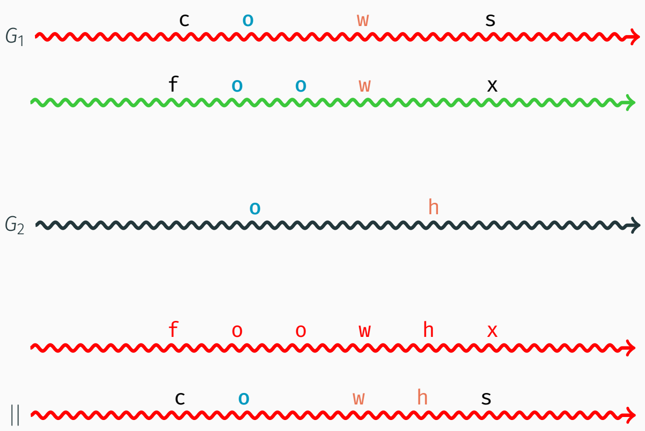
Verifying Opacity



Verifying Opacity



Verifying Opacity



Verifying Opacity

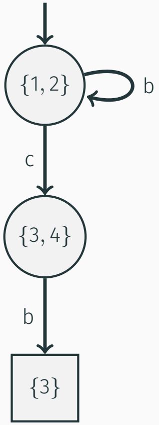
Opacity holds over composition when the attacker observes the interface, and we can verify opacity locally instead of doing it for the whole system.

Verifying Opacity

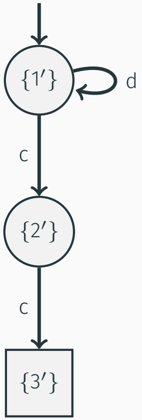
Opacity holds over composition when the attacker observes the interface, and we can verify opacity locally instead of doing it for the whole system.

- Check opacity of the local module’s secret
  - If opaque locally, it’s opaque for the composed system
  - Otherwise, iterate by composing the system with another module. If no other modules remain, it is not opaque for the composed system

Verifying Opacity



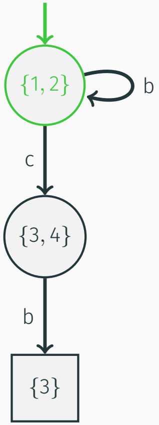
(a) $Det_{\Sigma_a}(G_1)$



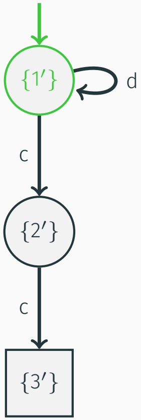
(b) $Det_{\Sigma_a}(G_2)$

(c) Full system

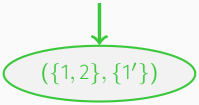
Verifying Opacity



(a) $Det_{\Sigma_a}(G_1)$

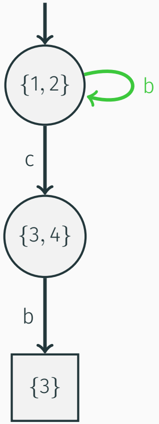


(b) $Det_{\Sigma_a}(G_2)$

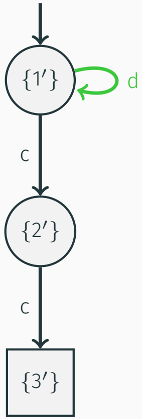


(c) Full system

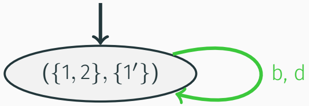
Verifying Opacity



(a) $Det_{\Sigma_a}(G_1)$



(b) $Det_{\Sigma_a}(G_2)$



(c) Full system

Verifying Opacity

$$\begin{tikzcd}
{} \arrow[d] \\
\{1,2\} \arrow[loop right, "b"] \arrow[d, "c"'] \\
\{3,4\} \arrow[d, "b"'] \\
\boxed{\{3\}}
\end{tikzcd}$$

(a) $Det_{\Sigma_a}(G_1)$

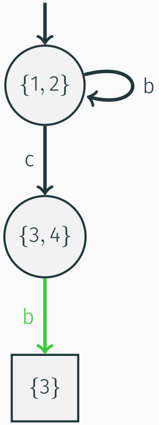
$$\begin{tikzcd}
{} \arrow[d] \\
\{1'\} \arrow[loop right, "d"] \arrow[d, "c"'] \\
\{2'\} \arrow[d, "c"'] \\
\boxed{\{3'\}}
\end{tikzcd}$$

(b) $Det_{\Sigma_a}(G_2)$

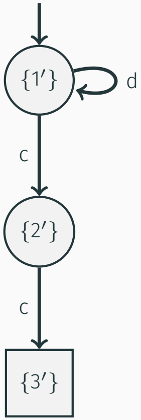
$$\begin{tikzcd}
{} \arrow[d] \\
(\{1,2\},\{1'\}) \arrow[loop right, "{b, d}"] \arrow[d, "c"'] \\
(\{3,4\},\{2'\})
\end{tikzcd}$$

(c) Full system

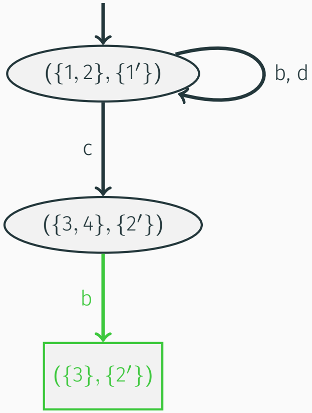
Verifying Opacity



(a) $Det_{\Sigma_a}(G_1)$



(b) $Det_{\Sigma_a}(G_2)$



(c) Full system

Enforcing Opacity in Modular Systems

Enforcing Opacity

- We can enforce opacity within each module separately to enforce it for the global system

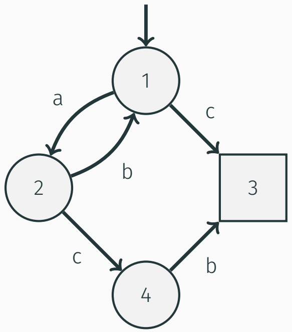
Enforcing Opacity

- We can enforce opacity within each module separately to enforce it for the global system
- Can observe some events

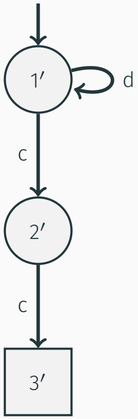
Enforcing Opacity

- We can enforce opacity within each module separately to enforce it for the global system
- Can observe some events
- Can control some events

Enforcing Opacity

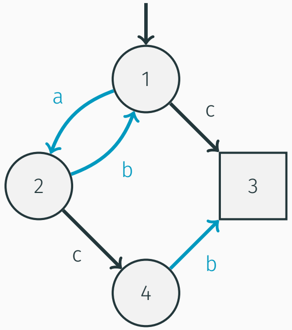


(a) G_1

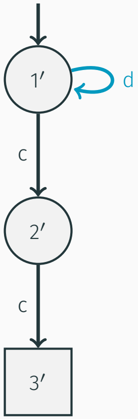


(b) G_2

Enforcing Opacity

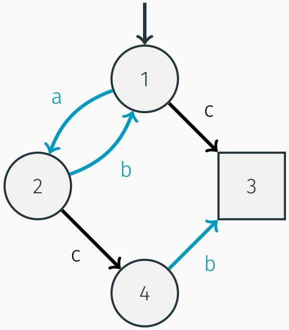


(a) G_1

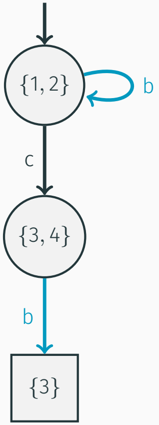


(b) G_2

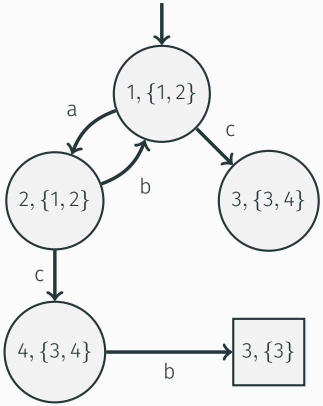
Enforcing Opacity



(a) G_1

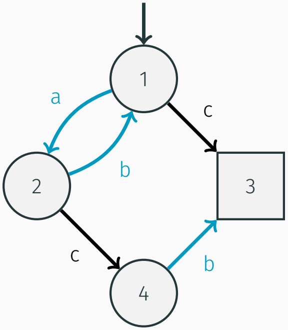


(b) $Det_{\Sigma_a}(G_1)$

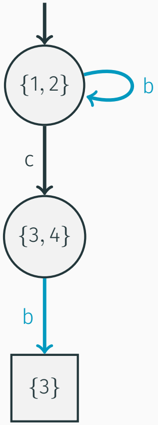


(c) Controller View

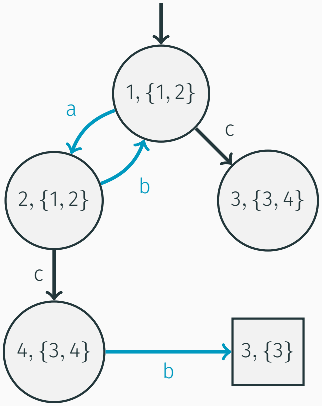
Enforcing Opacity



(a) G_1

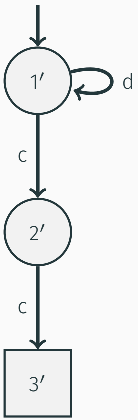


(b) $Det_{\Sigma_a}(G_1)$

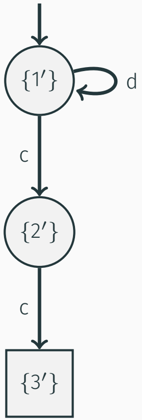


(c) Controller View

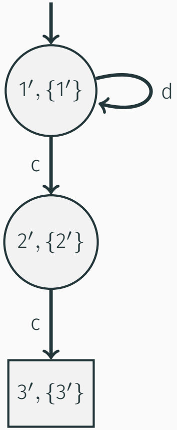
Enforcing Opacity



(a) G_2

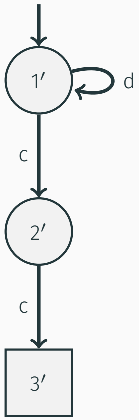


(b) $Det_{\Sigma_a}(G_2)$

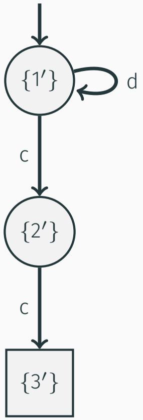


(c) Controller view

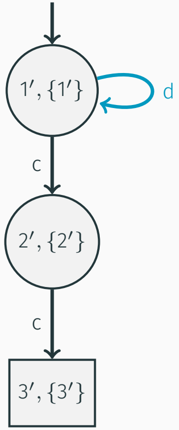
Enforcing Opacity



(a) G_2



(b) $Det_{\Sigma_a}(G_2)$

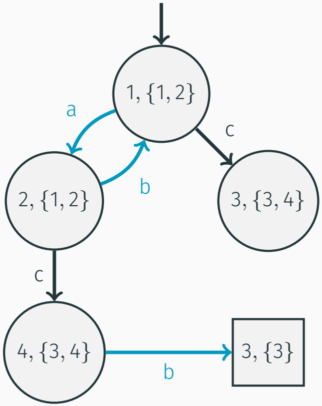


(c) Controller view

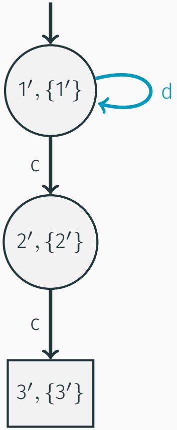
Enforcing Opacity

Naive approach: enforce opacity locally without considering other modules.

Enforcing Opacity

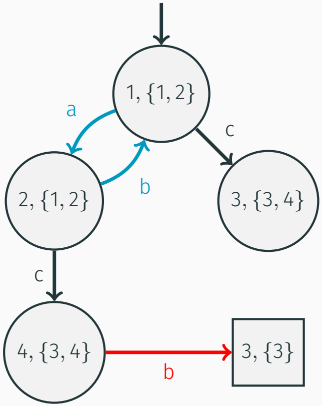


(a) $\mathcal{C}_1$

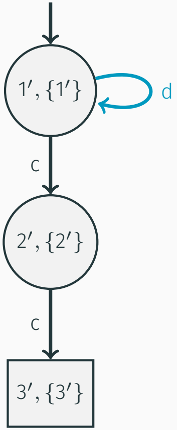


(b) $\mathcal{C}_2$

Enforcing Opacity

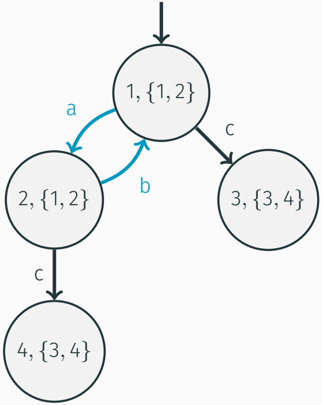


(a) $\mathcal{C}_1$

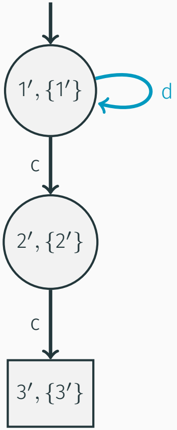


(b) $\mathcal{C}_2$

Enforcing Opacity

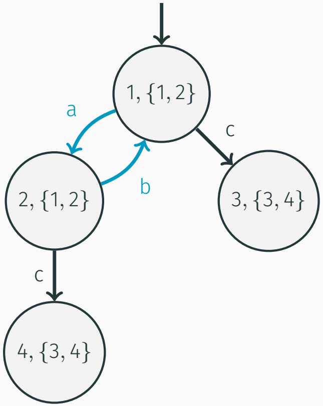


(a) $\mathcal{C}_1$

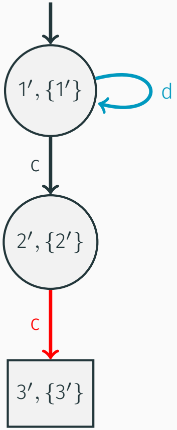


(b) $\mathcal{C}_2$

Enforcing Opacity

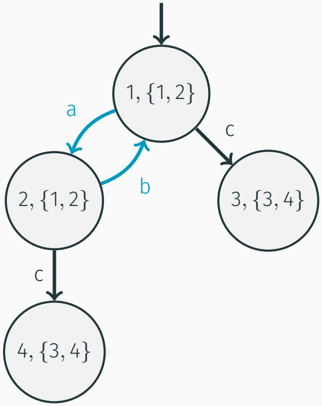


(a) $\mathcal{C}_1$

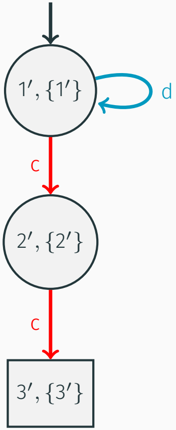


(b) $\mathcal{C}_2$

Enforcing Opacity

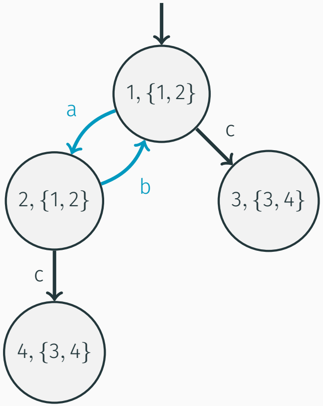


(a) $\mathcal{C}_1$

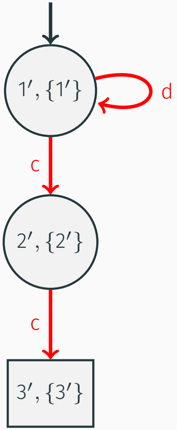


(b) $\mathcal{C}_2$

Enforcing Opacity

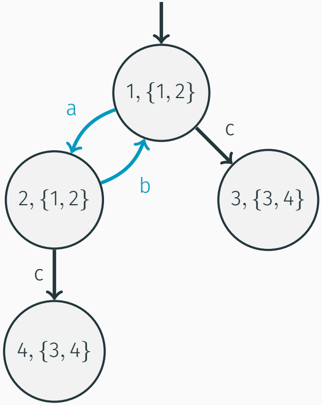


(a) $\mathcal{C}_1$



(b) $\mathcal{C}_2$

Enforcing Opacity



(a) $\mathcal{C}_1$

(b) $\mathcal{C}_2 = \emptyset$

Enforcing Opacity

Better approach:

- Verify if the secret is opaque as previously explained

Enforcing Opacity

Better approach:

- Verify if the secret is opaque as previously explained
    1. If opaque, use a fully permissive controller
    2. Otherwise, get the supremal controller of the composed system

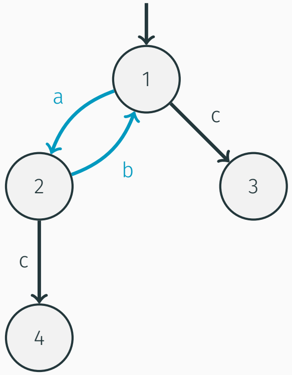
Enforcing Opacity

Better approach:

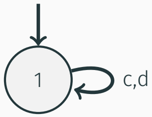
- Verify if the secret is opaque as previously explained
    1. If opaque, use a fully permissive controller
    2. Otherwise, get the supremal controller of the composed system

We observed that $\mathcal{S}_2$ was opaque, so we can use the fully permissive controller for G_2.

Enforcing Opacity



(a) $\mathcal{C}_1$



(b) $\mathcal{C}_2$

Enforcing Opacity

If we assume the attacker knows construction of only the system, **there is an optimal controller** if the controllable alphabet is a subset of the observable alphabet.

Enforcing Opacity

If we assume the attacker knows construction of only the system, **there is an optimal controller** if the controllable alphabet is a subset of the observable alphabet.

If we assume the attacker knows the system and the controller, a solution only exists when...

Enforcing Opacity

If we assume the attacker knows construction of only the system, **there is an optimal controller** if the controllable alphabet is a subset of the observable alphabet.

If we assume the attacker knows the system and the controller, a solution only exists when...

1. The controllable alphabet is a subset of the observable alphabet; and
2. The observable and attacker alphabets are comparable:
    2.1 The **observable alphabet is a subset** of the attacker's alphabet
    2.2 The **attacker's alphabet is a subset** of the observable alphabet

Conclusion

- Optimizations for verifying opacity in modular systems

Conclusion

- Optimizations for **verifying** opacity in modular systems
- New means of **enforcing** opacity in local modules of modular systems

Conclusion

- Optimizations for **verifying** opacity in modular systems
- New means of **enforcing** opacity in local modules of modular systems
- Currently, not possible when attacker knows the controller's architecture

Conclusion

Questions?