Hiding sensitive information from the scheduler

Kostas Chatzikokolakis

joint work with Catuscia Palamidessi

Why we need to hide information from the scheduler

- In security
  - Protocols often use randomization to obfuscate the link between the observable and the hidden events
  - Most of the times the outcome of the random choices must remain secret
- In our models (process calculi, automata)
  - The scheduler resolves the nondeterminism
  - It is assumed to have full knowledge of the state of the system
- Problem: the scheduler can leak the outcome of a prob. choice by depending its decisions on it

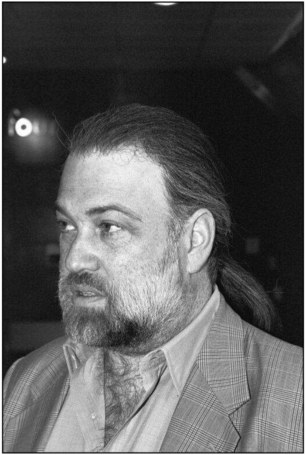
Why we need to hide information from the scheduler

- In security
  - Protocols often use randomization to obfuscate the link between the observable and the hidden events
  - Most of the times the outcome of the random choices must remain secret
- In our models (process calculi, automata)
  - The scheduler resolves the nondeterminism
  - It is assumed to have full knowledge of the state of the system
- Problem: the scheduler can leak the outcome of a prob. choice by depending its decisions on it

Why we need to hide information from the scheduler

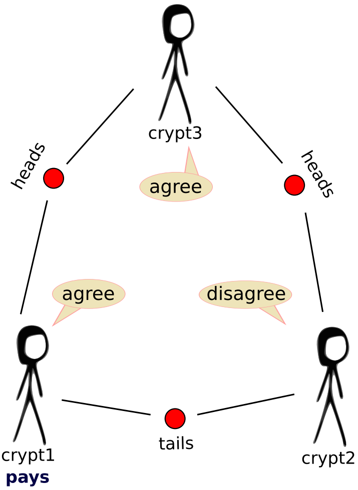
- In security
  - Protocols often use randomization to obfuscate the link between the observable and the hidden events
  - Most of the times the outcome of the random choices must remain secret
- In our models (process calculi, automata)
  - The scheduler resolves the nondeterminism
  - It is assumed to have full knowledge of the state of the system
- Problem: the scheduler can leak the outcome of a prob. choice by depending its decisions on it

Example: The dining cryptographers protocol

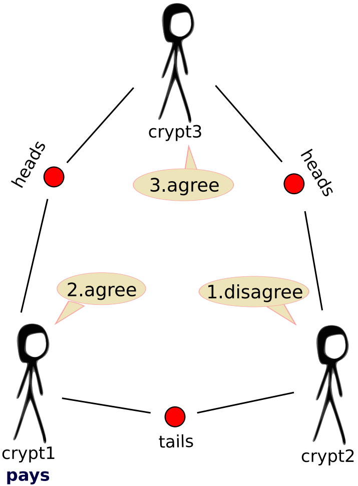


Who is this guy?

Example: The dining cryptographers protocol



Example: The dining cryptographers protocol



Formalizing strong anonymity

- Without non-determinism

$$p(aad \mid crypt_1) = p(aad \mid crypt_2)$$

- With non-determinism

$$p_S(aad \mid crypt_1) = p_S(aad \mid crypt_2) \quad \text{for all schedulers } S$$

- Take $S =$ scheduler who proritizes the payer

$$0 < p_S(a_1a_2d_3 \mid crypt_1) \neq p_S(a_1a_2d_3 \mid crypt_2) = 0$$

Formalizing strong anonymity

- Without non-determinism

$$p(aad \mid crypt_1) = p(aad \mid crypt_2)$$

- With non-determinism

$$p_S(aad \mid crypt_1) = p_S(aad \mid crypt_2) \quad \text{for all schedulers } S$$

- Take $S =$ scheduler who proritizes the payer

$$0 < p_S(a_1a_2d_3 \mid crypt_1) \neq p_S(a_1a_2d_3 \mid crypt_2) = 0$$

Formalizing strong anonymity

- Without non-determinism

$$p(aad \mid crypt_1) = p(aad \mid crypt_2)$$

- With non-determinism

$$p_S(a_1a_2d_3 \mid crypt_1) = p_S(a_1a_2d_3 \mid crypt_2) \quad \text{for all schedulers } S$$

- Take $S =$ scheduler who proritizes the payer

$$0 < p_S(a_1a_2d_3 \mid crypt_1) \neq p_S(a_1a_2d_3 \mid crypt_2) = 0$$

Formalizing strong anonymity

- Without non-determinism

$$p(aad \mid crypt_1) = p(aad \mid crypt_2)$$

- With non-determinism

$$p_S(a_1a_2d_3 \mid crypt_1) = p_S(a_1a_2d_3 \mid crypt_2) \quad \text{for all schedulers } S$$

- Take $S =$ scheduler who proritizes the payer

$$0 < p_S(a_1a_2d_3 \mid crypt_1) \neq p_S(a_1a_2d_3 \mid crypt_2) = 0$$

We need to restrict the scheduler

- Two views of this problem
  - Verification problem: we cannot verify this protocol
  - Security problem: realistic attacks can be based on the scheduler eg. the payer needs more time to compute the message to send
- We need to restrict the scheduler
- How to do that?

Task PIOAs

Canetti, Cheung, Kaynar, Liskov, Lynch, Pereira, Segala

Probabilistic Input/Output Automata

$$\begin{aligned} \mathcal{A} &= (Q, q_{\mathcal{A}}, I, O, H, D), \text{ where} \\ Q &- \text{set of states} \\ q_{\mathcal{A}} &- \text{start state} \\ I, O, H &- \text{pairwise disjoint sets of actions} \\ D &\subseteq Q \times Act \times Disc(Q) \end{aligned}$$

Satisfying transition determinism:

for all $q \in Q$ there is at most one transision labelled by a

Task PIOAs

- PIOA $+$ an equivalence relation R on $I \cup O$
- Task: an equivalence class of R
- Action determinism:

$$\text{for all } q \in Q \text{ and task } T$$
$$\text{there is at most one action } a \in T \text{ enabled in } q$$

- Task schedule: a (possibly infinite sequence) $T_1, T_2, \ldots$ of tasks
- Drawback: schedulers are oblivious

A process-algebraic approach

Goals and design features

- Fine-grained control: no unnecessary restrictions
- Keep our previous model, add annotations
- Use a simple language: CSS with internal probabilistic choice
- A process provides labels to the scheduler
- The scheduler can be seen as a (simple) process that runs in parallel to the main process and guides its execution

Syntax of CCS_σ

$P, Q ::=$		**processes**	$S, T ::=$	**scheduler**
	$L{:}\alpha.P$	prefix	$L.S$	single action
$\mid$	$P \mid Q$	parallel	$\mid \; (L, L).S$	synchronization
$\mid$	$P + Q$	non-determ.	$\mid$ **if** L **then** S **else** S	label test
$\mid$	$L{:}\sum_i p_i P_i$	prob. choice	$\mid \; 0$	nil
$\mid$	$(\nu a)P$	restriction		
$\mid$	$!P$	replication		
$\mid$	$L{:}0$	nil	$CP ::= P \parallel S$	**complete process**

Semantics by example

$$
\begin{array}{llll}
 & l:(l_1:a +_{\frac{1}{2}} l_2:\bar{b}) \mid l_3:c.(l_4:b + l_5:d) & \parallel & l.\textbf{if } l_1 \textbf{ then } \ldots \textbf{ else } l_3.(l_2, l_4) \\
\xrightarrow{\tau}_{\frac{1}{2}} & l_2:\bar{b} \mid l_3:c.(l_4:b + l_5:d) & \parallel & \textbf{if } l_1 \textbf{ then } \ldots \textbf{ else } l_3.(l_2, l_4) \\
\xrightarrow{c} & l_2:\bar{b} \mid (l_4:b + l_5:d) & \parallel & (l_2, l_4) \\
\xrightarrow{\tau} & 0 & \parallel & 0
\end{array}
$$

Expressiveness of the syntanctic scheduler

How powerful is the syntactic scheduler wrt the semantic one?

Linear labelling: all labels are disjoint

Proposition

Let $P_\sigma = P$ + a linear labelling. Then

$$\forall \zeta \; \exists S : \zeta([\![P]\!]) \sim [\![P_\sigma \parallel S]\!]$$

Non-linear labelings

- Non-linear labellings allow us to constrain the scheduler
- Example: $l\colon(l_1\colon a +_p l_2\colon b) \mid l_3\colon c \mid l_4\colon d$
- **Goal**: do the probabilistic choice. Then if a is available do c, otherwise do d

$$l.\mathbf{if}\ l_1\ \mathbf{then}\ l_3\ \mathbf{else}\ l_4$$

- However using the same label we can hide the outcome:

$$l\colon(l_1\colon a +_p l_1\colon b) \mid l_3\colon c \mid l_4\colon d$$

Private choice

Making all choices in the beginning should make no difference.

Theorem

$$C[l\!:\!\tau.P] +_p C[l\!:\!\tau.Q] \approx C[P +_p Q]$$

Key point: the labels of the context are duplicated

Also: $\approx$ is a congruence

Still a lot of work to be done

- Our understading of restricted schedulers is limited
  - What types of restrictions are needed
  - Other formalisms, comparisons
  - How do they affect compositionality
- What about model checking
  - How can the algorithms be adapted?
  - Tools that allow to express restrictions on the scheduler
  - Verify properties for individual schedulers

Thank you

Questions?