Using Task-Structured PIOAs to Analyze Cryptographic Protocols

Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, *Olivier Pereira* and Roberto Segala

FCC 2006

Motivation

Make:

- systematic proofs
- in a composable security setting
- considering probabilistic and nondeterministic behaviors
- including nondeterministic protocol specification

Nondeterministic behavior

Why? Experience from concurrency theory says:

- just specify what is needed for the protocol to work
- simplicity: avoids “clutter” in the specification
- generality: keeps freedom for the implementer

Example: Oblivious Transfer

OT functionality without internal nondeterminism:

Version 1:

- on input (x_0, x_1) from T, store (x_0, x_1)
- on input i from R: if input (x_0, x_1) was received, send x_i to R, else do nothing

Example: Oblivious Transfer

OT functionality without internal nondeterminism:
Version 2:

- on input (x_0, x_1) from T: if input i was received, send x_i to R, else store (x_0, x_1)
- on input i from R: if input (x_0, x_1) was received, send x_i to R, else store i

Example: Oblivious Transfer

OT functionality without internal nondeterminism:
Version 2:

- on input (x_0, x_1) from T: if input i was received, send x_i to R, else store (x_0, x_1)
- on input i from R: if input (x_0, x_1) was received, send x_i to R, else store i

OT functionality with internal nondeterminism:

- on input (x_0, x_1) from T, store (x_0, x_1)
- on input i from R, store i
- if (x_0, x_1) and i have been received, send x_i to R

Motivation

Make:

- systematic proofs
- in a composable setting
- exhibiting probabilistic and nondeterministic behaviors
- including in protocol specification

We want to prove security *for every way to resolve the nondeterminism*

This work...

In this work, we propose:

- a new model for the analysis of crypto protocols
  - protocols can have internal nondeterminism
  - enables simulation based security for nondeterministic systems
- an analysis of an Oblivious Transfer protocol [EGL85,GMW87] in our model

Starting Point

Our starting point is PIOAs [Seg95, LSV03], which are interacting, abstract, automata:

- state variables
- actions (input, output, internal)
- transitions: $(\mathit{state} \times \mathit{action}) \longrightarrow \mathrm{Disc}(\mathit{states}) \cup \perp$

Starting Point

Our starting point is PIOAs [Seg95, LSV03], which are interacting, abstract, automata:

- state variables
- actions (input, output, internal)
- transitions: $(\mathit{state} \times \mathit{action}) \longrightarrow \mathrm{Disc}(\mathit{states}) \cup \perp$

Low-level nondeterminism for output and internal actions

- not algorithmically resolved
- not resolved in the analyzed systems

How do we resolve this nondeterminism?

Resolving nondeteminism

- PIOAs use schedulers with full knowledge of current state — way too powerful!
- We introduce *tasks*, i.e.,
  - equivalence classes on actions, abstracting from state variables (ex: send message 1, select key, . . .)
  - given a task, at most one possible (probabilistic) action
- We introduce *task schedulers*: just sequences of tasks
- Execution: read first task, find and execute the enabled action (if there is one), go to next task, . . .

Indistinguishability

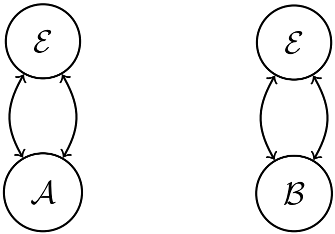
Implementation relation for task-PIOAs:

- $\mathcal{A} \leq \mathcal{B}$ means:

Indistinguishability

Implementation relation for task-PIOAs:

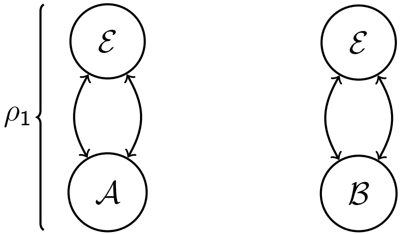
- $\mathcal{A} \leq \mathcal{B}$ means:
  $\forall$ environment $\mathcal{E}$ for $\mathcal{A}$ and $\mathcal{B}$,



Indistinguishability

Implementation relation for task-PIOAs:

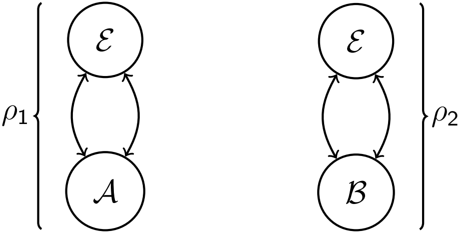
- $\mathcal{A} \leq \mathcal{B}$ means:
  $\forall$ environment $\mathcal{E}$ for $\mathcal{A}$ and $\mathcal{B}$,
  and $\forall$ task scheduler ρ_1 for $\mathcal{A}||\mathcal{E}$,



Indistinguishability

Implementation relation for task-PIOAs:

- $\mathcal{A} \leq \mathcal{B}$ means:
  $\forall$ environment $\mathcal{E}$ for $\mathcal{A}$ and $\mathcal{B}$,
  and $\forall$ task scheduler ρ_1 for $\mathcal{A}||\mathcal{E}$,
  $\exists$ task scheduler ρ_2 for $\mathcal{B}||\mathcal{E}$
  s.t. $\mathcal{E}$ cannot distinguish $\mathcal{A}$ from $\mathcal{B}$



Indistinguishability

Implementation relation for task-PIOAs:

- $\mathcal{A} \leq \mathcal{B}$ means:
  $\forall$ environment $\mathcal{E}$ for $\mathcal{A}$ and $\mathcal{B}$,
  and $\forall$ task scheduler ρ_1 for $\mathcal{A}||\mathcal{E}$,
  $\exists$ task scheduler ρ_2 for $\mathcal{B}||\mathcal{E}$
  s.t. $\mathcal{E}$ cannot distinguish $\mathcal{A}$ from $\mathcal{B}$

- Indistinguishability for nondeterminisitic systems

Computational Indistinguishability

Time-bounded Task-PIOAs:

- time-bound on the execution of each task
- bound on the length of the representation of all actions, state variables, ...

Computational Indistinguishability

Time-bounded Task-PIOAs:

- time-bound on the execution of each task
- bound on the length of the representation of all actions, state variables, ...

Approximate implementation relation for task-PIOAs:

- similar to the previous one, except:
  - time-bound on the environment
  - bound on the length of the task-schedulers
  - small probability of distinguishing allowed

Simulation Based Security

Simulation Based Security:

- Protocol π realizes functionality ϕ iff
  $\forall$ adversary task-PIOA $\mathcal{A}$, $\exists$ adversary task-PIOA $\mathcal{S}$:
  $\pi || \mathcal{A} \leq \phi || \mathcal{S}$

Simulation Based Security

Simulation Based Security:

- Protocol π realizes functionality ϕ iff
  $\forall$ adversary task-PIOA $\mathcal{A}$, $\exists$ adversary task-PIOA $\mathcal{S}$:
  $\pi||\mathcal{A} \leq \phi||\mathcal{S}$

Unwinding definition of $\leq$:

- Protocol π realizes functionality ϕ iff
  $\forall$ adversary task-PIOA $\mathcal{A}$, $\exists$ adversary task-PIOA $\mathcal{S}$:
  $\forall$ environment $\mathcal{E}$,
  $\forall$ task scheduler for $\pi||\mathcal{A}||\mathcal{E}$
  $\exists$ task scheduler for $\phi||\mathcal{S}||\mathcal{E}$:
  $\mathcal{E}$ cannot distinguish $\pi||\mathcal{A}$ from $\phi||\mathcal{S}$

Proving Security

Two variants of $\leq$:

- $\leq_0$, for perfect implementation
- $\leq_{neg,pt}$ for computational implementation

Proving Security

Two variants of $\leq$:

- $\leq_0$, for perfect implementation
- $\leq_{neg,pt}$ for computational implementation

$\leq_0$ proved using a sound simulation relation

- $\approx$ matching (distributions on) states
- very systematic proofs

Proving Security

Two variants of $\leq$:

- $\leq_0$, for perfect implementation
- $\leq_{neg,pt}$ for computational implementation

$\leq_0$ proved using a sound simulation relation

- $\approx$ matching (distributions on) states
- very systematic proofs

$\leq_{neg,pt}$ proved using computational assumptions

- Express computational assumptions as $C_1 \leq_{neg,pt} C_2$
- Composition: $C_1 \leq_{neg,pt} C_2 \Rightarrow C_1||Ifc \leq_{neg,pt} C_2||Ifc$

Proving Security

Two variants of $\leq$:

- $\leq_0$, for perfect indistinguishability
- $\leq_{neg,pt}$ for computational indistinguishability

Both these relations are:

- transitive: $\mathcal{A} \leq \mathcal{B}$ and $\mathcal{B} \leq \mathcal{C} \Rightarrow \mathcal{A} \leq \mathcal{C}$
- composable: $\mathcal{A} \leq \mathcal{B} \Rightarrow \mathcal{A}||\mathcal{C} \leq \mathcal{B}||\mathcal{C}$

Proving Security

Two variants of $\leq$:

- $\leq_0$, for perfect indistinguishability
- $\leq_{neg,pt}$ for computational indistinguishability

Both these relations are:

- transitive: $\mathcal{A} \leq \mathcal{B}$ and $\mathcal{B} \leq \mathcal{C} \Rightarrow \mathcal{A} \leq \mathcal{C}$
- composable: $\mathcal{A} \leq \mathcal{B} \Rightarrow \mathcal{A}||\mathcal{C} \leq \mathcal{B}||\mathcal{C}$

Modular proofs:

- $A \leq B$ proved as $A \leq A_1 \leq \cdots \leq A_n \leq B$
  - $\approx$ sequences of games, but for automata
- Composition properties allow reusing proofs for small systems in bigger ones

Example: Establishing $\mathcal{A} \leq_{neg,pt} \mathcal{B}$

Example: Hard-core predicates for trapdoor permutations

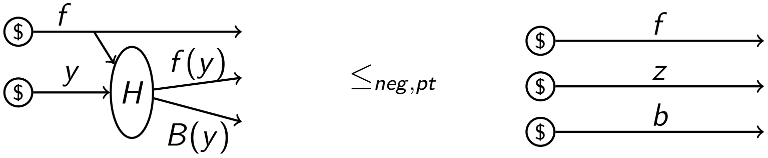
Crypto: B is a hardcore predicate for the Tdp family iff for every PPT Adv, there is a negligible ϵ:

$$\left| \begin{array}{l} \Pr[f \leftarrow Tdp; \\ \quad y \leftarrow Dom(Tdp); \\ \quad b \leftarrow B(y) : \\ \quad Adv(f, f(y), b) = 1] \end{array} - \begin{array}{l} \Pr[f \leftarrow Tdp; \\ \quad z \leftarrow Dom(Tdp); \\ \quad b \leftarrow \{0,1\} : \\ \quad Adv(f, z, b) = 1] \end{array} \right| \leq \epsilon$$

Defining H-C Predicates in terms of PIOAs

We transpose this classical crypto assumption to task-PIOAs.

$SH \leq_{neg,pt} SHR$:



SH SHR

Theorem: Crypto and task-PIOA formulations are equivalent!

Using Computational Assumptions

What's happening if we use 2 hard-core bits?

In some protocol, we:

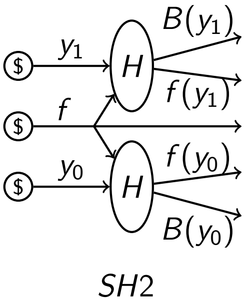
- select one trapdoor permutation f
- select two elements of the domain of f, say, (y_0, y_1)
- transmit $f(y_0), f(y_1), B(y_0), B(y_1)$

Do we keep the same indistinguishability guarantee?

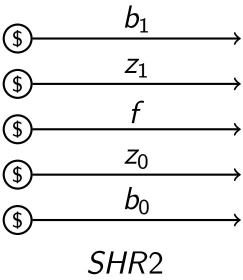
- that is, can $B(y_0)$ and $B(y_1)$ be distinguished from random bits?

Using our PIOAs Hardness Assumption

Our composition and transitivity properties allow proving $SH2 \leq_{neg,pt} SHR2$:

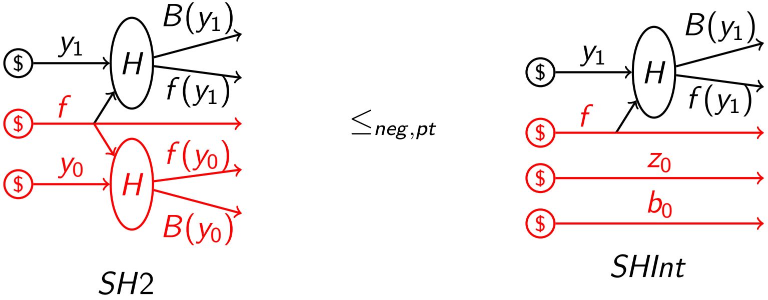


$\leq_{neg,pt}$



Using our PIOAs Hardness Assumption

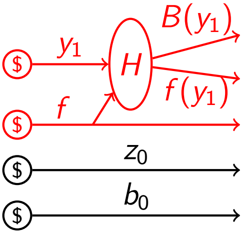
Consider the *SHInt* intermediate system. We have:



*SH*2 and *SHInt* are just *SH* and *SHR* composed with the same systems!

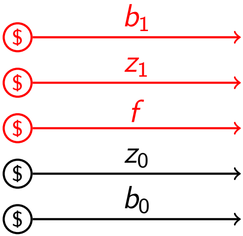
Using our PIOAs Hardness Assumption

We also have:



SHInt

$\leq_{neg,pt}$



SHR2

$SH2 \leq_{neg,pt} SHR2$ follows from our transitivity result!

Conclusion

Case-study on an Oblivious Transfer protocol [GMW87]
available: MIT-CSAIL-TR-2006-047, June. 2006.

We hope task-PIOAs provide a framework for:

- General, expressive, protocol specifications
- General, systematic, security proofs

Conclusion

Future works:

- General theorem for secure protocol composition in this model
- More general nondeterministic scheduling resolved at runtime
- Deal with other computational assumptions
- New case studies (key exchange, . . .)
- Mechanization
- . . .

Example: Needham-Schroeder-Lowe

Receiver role:

Version 1:

1. Receive $\{\!|N_a, A|\!\}_{K_B}$
2. Select N_b
3. Send $\{\!|N_a, N_b, B|\!\}_{K_A}$

Example: Needham-Schroeder-Lowe

Receiver role:

Version 1:

1. Receive $\{\!|N_a, A|\!\}_{K_B}$
2. Select N_b
3. Send $\{\!|N_a, N_b, B|\!\}_{K_A}$

Version 2:

1. Select N_b
2. Receive $\{\!|N_a, A|\!\}_{K_B}$
3. Send $\{\!|N_a, N_b, B|\!\}_{K_A}$

Example: Needham-Schroeder-Lowe

Receiver role:

Version 1:

1. Receive $\{|N_a, A|\}_{K_B}$
2. Select N_b
3. Send $\{|N_a, N_b, B|\}_{K_A}$

Version 2:

1. Select N_b
2. Receive $\{|N_a, A|\}_{K_B}$
3. Send $\{|N_a, N_b, B|\}_{K_A}$

- from a security point of view: who cares?
- according to the hardware, one solution might be better than the other