Computational Indistinguishability Logic
(Work in Progress)

G. Barthe$^1$ M. Daubignard$^2$ B. Kapron$^3$ Y. Lakhnech$^2$

$^1$IMDEA Software, Madrid $^2$University of Grenoble, CNRS - VERIMAG
$^3$University of Victoria

April 16, 2010
Introduction

Our goal:
Enable *Computer-aided verification of cryptographic constructions in the computational model.*
Ideally:

▶ For protocols and cryptographic primitives.
▶ In the concrete security framework.
▶ As automated as possible.
Two approaches

For security protocols:

▶ Indirect approach:
  ▶ A proof in a symbolic model.
  ▶ A computational soundness proof.

▶ Direct approach: [Cryptoverif, Ceticrypt]
  ▶ Security definitions and models are in the computational interpretation.
  ▶ Proofs are reductionist.
Indirect approach - advantages and drawbacks

Main advantages are:

▶ the level of abstraction,
▶ automated tools,

Drawbacks:

▶ The model has to be repeatedly adapted to guarantee computational soundness - new primitive → new soundness proof.
▶ The decidability results and verification procedures are to be revisited, extended and often are difficult to obtain.
▶ Unlikely to get reasonable concrete security bounds.
Our approach

A direct approach based on a logic (a formal system) for indistinguishability and event probabilities.

▶ Capture the main (may be eventually all) reasoning schema applied in cryptographic proofs.

▶ Explicitly distinguish between information-theoretic reasoning (no security loss) and reduction-based reasoning.

▶ Language-independent, i.e., can be instantiated by different languages to describe the system.

▶ Amenable to automation and proof certification.
Our starting point is a logic proposed by R. Impagliazzo and B. Kapron for indistinguishability [FOCS’03]. It is extended in several ways:

- Oracles: reasoning about oracles is crucial for many cryptographic constructions and protocols.
- Rules to reason about probabilities of events.
- A direct interpretation and soundness proof.
- Concrete security proofs.

It is also related to Yu Zhang’s work based on Hofmann’s Characterization of poly-time computations.
Oracle Systems

A system is described by:

- A (countable) set of memories with an initial memory $\bar{m}$.
- A set of oracles, where each oracle is given by a name and an implementation:
  \[
  O_o : \text{In}(o) \times M_o \rightarrow \mathcal{D}(\text{Out}(o) \times M_o)
  \]

- Distinguished oracles $O_{init}$ for initialization and $O_{final}$ for finalization, such that $\text{In}(O_{init}) = \text{Out}(O_{final}) = 1$. 
An adversary $A$ (for an oracle system $O$) is given by:

- a (countable) set $M_a$ of adversary memories and an initial memory $\tilde{m}_a$;
- a transition function, and an update function:

\[
\begin{align*}
A & : M_a \to \mathcal{D}(\text{Que} \times M_a) \\
A_\downarrow & : M_a \times \text{QuAn} \to \mathcal{D}(M_a)
\end{align*}
\]

Que: the set of queries, a query is an oracle name and an input; and Ans is the union of all $\text{Out}(o)$. 

**Oracle System Adversaries**
The composition $\mathcal{A} \mid \mathcal{O}$ of adversary $\mathcal{A}$ and oracle system $\mathcal{O}$ is a \textit{probabilistic transition system}:

$$\text{step}_{\mathcal{A} \mid \mathcal{O}}(m_a, m_o) \overset{\text{def}}{=} \begin{cases} \text{let } ((o, q), m'_a) \leftarrow \mathcal{A}(m_a) \text{ in} \\ \text{let } (a, m'_o) \leftarrow \mathcal{O}_o(q, m_o) \text{ in} \\ \text{let } m''_a \leftarrow \mathcal{A}_\downarrow(m'_a, (o, q, a)) \text{ in} \\ \text{return } ((o, q, a), (m''_a, m'_o)) \end{cases}$$

Only adversaries with bounded running time.
Executions and Traces

- Executions:

\[(m^0_a, m^0_o) \xrightarrow{\langle o_1, q_1, a_1 \rangle} (m^1_a, m^1_o) \xrightarrow{\langle o_2, q_2, a_2 \rangle} \ldots \xrightarrow{\langle o_k, q_k, a_k \rangle} (m^k_a, m^k_o)\]

such that \(o_1 = \mathcal{O}_{\text{init}}\) and \(o_k = \mathcal{O}_{\text{final}}\).

- Traces are obtained by deleting the adversary’s state:

\[
m^0_o \xrightarrow{\langle o_1, q_1, a_1 \rangle} m^1_o \xrightarrow{\langle o_2, q_2, a_2 \rangle} \ldots \xrightarrow{\langle o_k, q_k, a_k \rangle} m^k_o
\]

\[
\Pr[A \mid \mathcal{O} = \eta] = \prod_{i=0}^{k-1} \Pr[\text{step}_A \mid \mathcal{O}](m^i_a, m^i_o) = (\langle o_i, q_i, a_i \rangle, (m^i_{a+1}, m^i_{o+1})))
\]

\[
\Pr[A \mid \mathcal{O} : \tau] = \Pr[A \mid \mathcal{O} : T^{-1}(\tau)]
\]
Statements of the logic

- **Indistinguishability**: \( \emptyset \sim_{\epsilon} \emptyset' \)

  \[ \models \emptyset \sim_{\epsilon} \emptyset' \]

  if for any adversary,

  \[ | \Pr[A \mid \emptyset : r = 1] - \Pr[A \mid \emptyset' : r = 1] | \leq \epsilon \]

- **Event probability**: An event \( E \) is a mapping from traces to Bool. \( \emptyset :_{\epsilon} E \):

  \[ \models \emptyset :_{\epsilon} E \]

  if for any adversary,

  \[ \Pr[A \mid \emptyset : E] \leq \epsilon \]
Reasoning in CIL consists in deriving a statement from a set of hypotheses that capture cryptographic assumptions by applying derivation rules.

The rules may have CIL statements as premisses or external statements:

- Valid logic formulae, e.g. \( A \Rightarrow A \lor B \)
- Hoare triples for simple probabilistic programs.
- Equivalence of simple probabilistic programs.

There are two types of rules:

- Rules that do not rely on reduction arguments and do not cause loss in security.
- Rules that rely (their soundness is by) reduction and cause loss in security properties.
Derivation Rules

The first set of rules supports equational and external reasoning:

\[
\begin{align*}
\frac{}{\emptyset \sim_0 \emptyset} & & \frac{}{\emptyset \sim_\epsilon \emptyset'} & & \frac{}{\emptyset \sim_\epsilon \emptyset'} \sim_\epsilon \emptyset''
\end{align*}
\]

\[
\begin{align*}
\emptyset : \epsilon_i E_i \ (i \in I) & & E \Rightarrow \bigvee_{i \in I} E_i & & \text{UR}
\end{align*}
\]

\[
\begin{align*}
\forall o, \{\text{true}\} O\{\text{PR}[\varphi] \leq \epsilon_o\} & & \emptyset : \epsilon \diamond \varphi & & \text{FAIL}
\end{align*}
\]

For \( \varphi : \text{QuAn} \times M \times M \rightarrow \text{Bool} \) and \( \epsilon = \sum_{o \in N_o} k_o \epsilon_o \).
Context

An \(\mathcal{O}\)-context \(\mathcal{C}\) is given by:

- sets \(M_c\) of context memories, an initial memory \(\bar{m}_c\) and \(N_c\) of procedures;
- for every \(c \in N_c\), a query domain \(\text{In}_c(c)\), an answer domain \(\text{Out}_c(c)\), and forward and backwards implementations

\[
\begin{align*}
\mathcal{C}_c & : \text{In}_c(c) \times M_c \rightarrow \mathcal{D}(\text{Que} \times M_c) \\
\mathcal{C}_c & : \text{In}_c(c) \times \text{QuAn} \times M_c \rightarrow \mathcal{D}(\text{Out}_c(c) \times M_c)
\end{align*}
\]

- distinguished initial and finalization procedures \(c_i\) and \(c_f\).
Context Application

The application of an $\emptyset$-context $\mathbb{C}$ to $\emptyset$ defines an oracle system $\mathbb{C}[\emptyset]$ such that:

- the set of memories is $M_c \times M_o$, and the initial memory is $(\bar{m}_c, \bar{m}_o)$;
- the oracles are the procedures of $\mathbb{C}$, and their query and answer domains are given by $\mathbb{C}$. The initialization and finalization oracles are the initialization and finalization procedures of $\mathbb{C}$;
- the implementation of an oracle $c$ is:

$$
\lambda q_c, (m_c, m_o).
\quad \text{let } ((o, q_o), m'_c) \leftarrow \mathbb{C}_c \rightarrow (q_c, m_c) \text{ in}
\quad \text{let } (a_o, m'_o) \leftarrow \mathbb{O}_o(q_o, m_o) \text{ in}
\quad \text{let } (a_c, m''_c) \leftarrow \mathbb{C}_c \leftarrow (q_c, (o, q_o, a_o), m'_c) \text{ in}
\quad \text{return } (a_c, (m''_c, m'_o))
$$
Composition of Adversary with Context

- the set of memories is $M_c \times M_a \times \text{Que}_c$, and the initial memory is $(\bar{m}_c, \bar{m}_a, -)$;
- the transition function is:

$$
\lambda(m_c, m_a, -).
\quad \text{let } ((c, q_c), m'_a) \leftarrow A(m_a) \text{ in}
\quad \text{let } ((o, q), m'_c) \leftarrow C_c \rightarrow (q_c, m_c) \text{ in}
\quad \quad \text{return } ((o, q), (m'_c, m'_a, (o, q)))
$$

- the update function is:

$$
\lambda((m_c, m_a, (o_c, q_c)), (o_o, q_o, a_o)).
\quad \text{let } (a_c, m'_c) \leftarrow C_c \leftarrow (q_c, (o_o, q_o, a_o), m_c) \text{ in}
\quad (m'_c, A \downarrow (m_a, (o_c, q_c, a_c)), -)
$$
**SUB and NegSUG Rules**

\[
\frac{\mathcal{O} \sim_\epsilon \mathcal{O}'}{\mathcal{C}[\mathcal{O}] \sim_\epsilon \mathcal{C}[\mathcal{O}']} \quad \text{SUB} \quad \frac{\mathcal{O} :_\epsilon E \circ \mathcal{C}}{\mathcal{C}[\mathcal{O}] :_\epsilon E} \quad \text{NegSUB}
\]

The \( \mathcal{O} \)-event \( E \circ \mathcal{C} \) is defined as:

\[
\lambda \tau. \exists \tau^{\text{mix}}. \pi_{\mathcal{O}}(\tau^{\text{mix}}) = \tau \land E(\pi_{\mathcal{C}[\mathcal{O}]}(\tau^{\text{mix}}))
\]

**Proposition**

*Let \( \mathcal{O}, \mathcal{O}' \) be compatible oracle systems and \( \mathcal{C} \) be an \( \mathcal{O} \)-context.*

- *If \( \mathcal{C} \) is an indistinguishability context and \( \models \mathcal{O} \sim_\epsilon \mathcal{O}' \) then \( \models \mathcal{C}[\mathcal{O}] \sim_\epsilon \mathcal{C}[\mathcal{O}'] \).*

- *For every \( \mathcal{C}[\mathcal{O}] \)-event \( E \), if \( \models \mathcal{O} :_\epsilon E \circ \mathcal{C} \) then \( \models \mathcal{C}[\mathcal{O}] :_\epsilon E \).*
Equivalence of Oracle Systems

- Often, one needs to replace an oracle system by (an almost) equivalent one.
- Two probabilistic transition systems are equivalent, if they associate the same probability to each event.
- Two oracle systems $O$ and $O'$ are called *equivalent*, if for any adversary $A$, the underlying probabilistic transition systems $A \mid O$ and $A \mid O'$ are equivalent.
- Bisimulations provide a powerful method for proving equivalence of probabilistic transition systems.
- We lift bisimulation-based reasoning to oracle systems. More specifically, we define a collection of conditions that ensure the existence of a bisimulation between $A \mid O$ and $A \mid O'$, for every $A$, and hence, the equivalence of $O$ and $O'$. 
Almost Equivalence of Oracle Systems

- Almost equivalence of oracle systems: this is when equivalence holds as long as the adversary does not do something bad.
- The bad is specified by:
  \[ \varphi \subseteq \text{Que} \times \text{Ans} \times (M_o + M'_o) \times (M_o + M'_o) \]
- Equivalence is ensured by: existence of an equivalence relation \( R \subseteq (M_o + M'_o) \times (M_o + M'_o) \) on memories that satisfies
  - **Initialization:** \( \bar{m}_o R \bar{m}_a \)
  - **stability:** if \( m'_1 R m'_2 \) then
    \[ \varphi((o, q, a), m_1, m'_1) \iff \varphi((o, q, a), m_2, m'_2) \]
  - **compatibility:** if \( \varphi((o, q, a), m_1, m'_1) \), then
    \[ \Pr[\hat{O}_o(q, m_1) \in (a, C)] = \Pr[\hat{O}_o(q, m_2) \in (a, C)] \]
    where \( C \) is the equivalence class of \( m'_1 \) under \( R \).

Notation: \( \emptyset \equiv_{R, \varphi} \emptyset' \)

Proposition

*If \( \emptyset \equiv_{R, \varphi} \emptyset' \) then \( A | \emptyset \sim_{\neg \varphi} A | \emptyset' \).*
Oracle Rules

\[
\begin{align*}
\text{NegOR}\forall & : \\
\tau : \epsilon & (E \land \Box(\varphi)) \quad \tau \equiv_{R,\varphi} \tau' & E \in R^E \\
\tau' : \epsilon & (E \land \Box(\varphi)) \\
\end{align*}
\]

\[
\begin{align*}
\text{Neg\Box} & : \\
\tau : \epsilon & \Diamond \neg \varphi \quad \tau \equiv_{R,\varphi} \tau' & \tau' : \epsilon & \Diamond \neg \varphi \\
\end{align*}
\]

\[
\begin{align*}
\text{OR} & : \\
\tau : \epsilon & \Diamond \neg \varphi \quad \tau \equiv_{R,\varphi} \tau' & \tau \sim_{\epsilon} \tau' \\
\end{align*}
\]

\[
\begin{align*}
\text{NegOR}\exists & : \\
\tau : \epsilon & (E \land \Box(\varphi)) \cup \neg \varphi \quad \tau \equiv_{R,\varphi} \tau' & E \in R^E \\
\tau' : \epsilon & (E \land \Box(\varphi)) \cup \neg \varphi \\
\end{align*}
\]

**Proposition**

*For all oracle systems \(\tau\) and \(\tau'\) and testable \(\varphi\) such that \(\tau \equiv_{R,\varphi} \tau'\), every \(R\)-compatible events \(E\), and adversary \(A\):*

- \(P_R(A \mid \tau : E \land \Box(\varphi)) = P_R(A \mid \tau' : E \land \Box(\varphi))\)
- \(P_R(A \mid \tau : (E \land \Box(\varphi)) \cup \neg \varphi) = P_R(A \mid \tau : (E \land \Box(\varphi)) \cup \neg \varphi)\)
Early-late sampling and Determinization

Bisimulation is in many cases too strong, in particular for moving samples earlier (later) in the execution.

\[ pq = \sum p_i x_i \]
Rules based on Deterinization

\[ \emptyset' : \epsilon \ E \circ \pi_{M_0} \quad \text{NegDET} \]

\[ \emptyset : \epsilon \ E \]

\[ \emptyset \leq_{\text{det}(\gamma)} \emptyset' \quad \text{DET} \]

\[ \emptyset \sim_0 \emptyset' \]
Summary of the Rules

(REF), (SYM), (TRAN), (UR), (FAIL)

\[
\begin{align*}
\emptyset \sim_\epsilon \emptyset' & \quad \text{SUB} \\
\emptyset \sim_\epsilon C[\emptyset] & \quad \overline{\text{C[\emptyset]}} \sim_\epsilon C[\emptyset'] \\
\emptyset' : \epsilon E \circ \pi_{M_o} & \quad \text{NegDET} \\
\emptyset : \epsilon E & \\
\emptyset' : \epsilon \overline{\diamond \neg \phi} & \quad \text{Neg} \overline{\diamond} \\
\emptyset : \epsilon \square(\phi) & \quad E \in R^E \\
\emptyset' : \epsilon \square(\phi) & \\
\emptyset : \epsilon \overline{\neg \phi} & \quad \emptyset \equiv_{R, \varphi} \emptyset' \\
\emptyset : \epsilon (E \land \square(\phi))U
\overline{\neg \varphi} & \quad \text{NegOR} \exists \\
\emptyset' : \epsilon (E \land \square(\phi))U \overline{\neg \varphi} & \\
\emptyset : \epsilon E \circ C & \quad \overline{\text{C[\emptyset]}} : \epsilon E \\
\emptyset : \epsilon \overline{\neg \phi} & \quad \emptyset \equiv_{R, \varphi} \emptyset' \\
\emptyset : \epsilon (E \land \square(\phi))U \overline{\neg \varphi} & \quad E \in R^E \varphi \text{ testable} \\
\emptyset : \epsilon (E \land \square(\phi))U \overline{\neg \varphi} & \quad \emptyset \equiv_{R, \varphi} \emptyset' \\
\emptyset : \epsilon (E \land \square(\phi))U \overline{\neg \varphi} & \quad E \in R^E \varphi \text{ testable}
\end{align*}
\]
Let $\mathcal{O}$ be an oracle system with oracle names $o_1, \ldots, o_n$. Then, in $\mathcal{O} : \epsilon E$, $\epsilon$ is a function:

$$\epsilon : \mathbb{N}^{n+1} \rightarrow [0, 1]$$

and its interpretation is:
For every adversary $A$ that calls $o_i$ at most $q_i$-times and terminates in time $t + \sum_i q_i$,

$$\Pr[A \mid \mathcal{O} : E] \leq \epsilon(t, q_1, \ldots, q_n).$$

- “$A$ calls $o_i$ at most $q_i$-times...” means

$$\Pr[A \mid \mathcal{O} = \eta : \#\langle o_i, -, - \rangle \leq q_i] = 1$$
NegSUB revisited

\[ \emptyset :_{\epsilon} E \circ C \quad \text{NegSUB} \]

\[ C[\emptyset] :_{\epsilon_{\text{C}}} E \]

Assume \( C \) has procedures \( c_1, \ldots, c_m \). Define
\( \alpha : \{ o_1, \ldots, o_n \} \times \{ c_1, \ldots, c_m \} \rightarrow \{ 0, 1 \} \) such that \( \alpha(o_i, c_j) = 1 \) iff \( c_j \) may call \( o_i \), i.e.

\[ \exists m_c \in M_c \exists q \in \text{In}_c(c_j). \quad \sum_{m'_c \in M_c} \text{PR}[C \rightarrow (c_j)(m_c, q) = (o_i, m'_c)] > 0 \]

Then,

\[ \epsilon_c(t, q'_1, \ldots, q'_m) = \epsilon(t + T_C, \sum_{j=1}^{m} \alpha(o_1, c_j)q'_j, \ldots, \sum_{j=1}^{m} \alpha(o_n, c_j)q'_j) \]

where \( T_C = \sum_{j+1}^{m} q'_j(T_C \rightarrow (c_j) + T_C \leftarrow (c_j)) \).
Probabilistic Signature Scheme

Bellare&Rogaway’96.

- A signature scheme based on one-way trapdoor permutations.
- Part of the PKCS standard.

Informal description:

- The scheme uses 3 hash functions in the random oracle model.
- The signature of $M$ is

$$f^{-1}(H(M|r)|G(H(M|r)) \oplus r|F(H(M|r)))$$

where $r$ is sampled in $\{0,1\}^{k_1}$, $H : \{0,1\}^{km+k_1} \rightarrow \{0,1\}^{k_2}$, $G : \{0,1\}^{k_2} \rightarrow \{0,1\}^{k_1}$ and $F : \{0,1\}^{k_2} \rightarrow \{0,1\}^{k_0}$.

- To verify $(M,x)$, let $y = f(x)$, $w_1|w_2|w_3 = y$, $r = G(w_1) \oplus w_2$ and verify

$$w_1 = H(M|r) \land w_3 = F(w_1)$$
PSS as Oracle system

- $\mathcal{O}_{\text{init}}$ samples $(f, f^{-1})$ and returns $(f, (f, f^{-1}, L_H : [], L_F : [], L_G : []))$.

- $\mathcal{O}_H : \lambda x, m. \ (x, r) \in L_H \rightarrow \text{return} \ (r, m)$
  \[ \top \rightarrow \text{let} \ r \leftarrow \{0, 1\}^{k_2} \text{ in} \]
  \[ \text{return} \ (r, m[L_H := L_H :: (x, r)]) \]

- Similarly for $\mathcal{O}_F$ and $\mathcal{O}_G$.

- $\mathcal{O}_S : \lambda M, m. \ \text{let} \ r \leftarrow \{0, 1\}^{k_1} \text{ in} \]
  \[ \text{let} \ (w_1, m_1) \leftarrow \mathcal{O}_H(M|r, m) \text{ in} \]
  \[ \text{let} \ (w_2, m_2) \leftarrow \mathcal{O}_G(w_1, m_1) \text{ in} \]
  \[ \text{let} \ (w_3, m_3) \leftarrow \mathcal{O}_F(w_1, m_2) \text{ in} \]
  \[ \text{return} \ (f^{-1}(w_1|w_2 \oplus r|w_3), m_3) \]
The Security Property

An adversary breaks PSS, if she is able to provide a message and a signature of that message that has not been produced by $O_S$.

$$EF\text{-CMA} = \Diamond[(\langle O\text{\_final}, R_1\| R_2, 1 \rangle, m_1, m_2) \land V(R_1, R_2, m_2)] \land \Box\neg(\langle O_S, R_1, R_2 \rangle, -, -)$$

$$V : \lambda M, x, m. \text{ let } w_1\| w_2\| w_3 = f(x) \text{ in}$$
$$\text{ let } (w_1, g) \in m.L_G \text{ in}$$
$$\text{ let } r = w_2 \oplus g \text{ in}$$
$$\text{ return } (M\| r, w_1) \in m.L_H \land (w_1, w_3) \in m.L_F$$

Notice that $r$ is uniquely defined. Denote it $r(R_2)$. 
One-way Permutation

- Memory: \((y, pk, sk, b)\), \(y \in \{0, 1\}^k\), \(pk, sk\) public and secret keys of the One-way permutation. Initial memory \(b = \text{true}\).

- \(\mathcal{O}_{\text{init}}\):

\[
\lambda x, m \quad b \rightarrow \text{let } y \leftarrow \{0, 1\}^k \text{ in} \\
\text{let } (pk, sk) \leftarrow \mathcal{K} \text{ in} \\
\text{return } ((pk, y), (y, pk, sk, \text{false})) \\
\top \rightarrow ((pk, y), (y, pk, sk, \text{false}))
\]

- \(\mathcal{O}_{\text{final}}\): \(\lambda x, m. \text{return } (1, m)\)

- \(V_{\text{ow}}\): \(\lambda \tau, m. \quad \text{let } \tau = \tau' \langle \mathcal{O}_{\text{final}}; x, 1 \rangle \rightarrow m \text{ in} \\
f(m.pk, x) = m.y\)

- \(\emptyset_{\text{ow}} : \epsilon_{\text{ow}(t)} V_{\text{ow}}\)
Two cases

\[ \text{EF-CMA} = \text{EF-CMA}_1 \lor \text{EF-CMA}_2 \]

where

\[ \text{EF-CMA}_1 = \text{EF-CMA}_1 \land R_1 | r(R_2) \notin L_H \]
\[ \text{EF-CMA}_2 = \text{EF-CMA}_1 \land R_1 | r(R_2) \in L_H \]

(UR)

\[ \frac{\text{PSS} : 2^{-k_2} \text{ EF-CMA}_1 \quad \text{PSS} : \epsilon 2^{-k_2} \text{ EF-CMA}_2}{\text{PSS} : \epsilon \text{ EF-CMA}} \]
PSS : $2^{-k_2}$ EF-CMA$_1$

\[
\emptyset = \text{O}_\text{init}, \text{O}_H, \text{O}_\text{final}
\]
\[
\text{O}_\text{O}_\text{init} : \lambda x, m. (1, m[L_H := []])
\]
\[
\text{O}_\text{O}_H : \lambda x, m. (x, r) \in L_H \rightarrow \text{return} (r, m)
\]
\[
\top \rightarrow \text{let } r \leftarrow \{0, 1\}^{k_2} \text{ in}
\]
\[
\text{return} (r, m[L_H := L_H :: (x, r)])
\]
\[
\text{O}_\text{O}_\text{final} = \text{O}_\text{O}_H
\]

\[
\text{GuessHash} = \Diamond \left[ \left( \langle \text{O}_\text{final}, R_1 | R_2, 1 \rangle, m_1, m_2 \right) \land m_2.L_H(R_1) = R_2 \right] \land R_1 \not\in m_1.L_H
\]

\[
\begin{align*}
\text{FAIL} \\
\emptyset_H : 2^{-k_2} \text{ GuessHash} \\
\text{C}[\emptyset_H] : 2^{-k_2} \text{ EF-CMA}_1
\end{align*}
\]

\[
\text{C-memory: } (f, f^{-1}, L_F : [], L_G : [])
\]
PSS1

Compute $F(h)$ and $G(h)$ as soon as $h$ is produced by $H$. To keep the system consistent with PSS, we need some book keeping using lists $L'_F$ and $L'_G$.

$$O_{\mathcal{O}_G} : \lambda x, m. \begin{cases} (x, w_2) \in L_G & \rightarrow (w_2, m) \\ (x, w_2) \in L'_G & \rightarrow (w_2, m[L'_G := L'_G \setminus (x, w_2)], \\ & \quad L_G := L_G :: (x, w_2)) \end{cases}$$

$$\top \rightarrow \text{let } w_2 \leftarrow \{0, 1\}^{k_1} \text{ in}$$

$$\quad (w_2, m[L_G := L_G :: (x, w_2)])$$

return $(w_2, m_3)$

$$O_{\mathcal{O}_H} : \lambda x, m. \begin{cases} \text{let } (w_1, m_1) \leftarrow O_H(x, m) \text{ in} \\ \text{let } (w_2, m_2) \leftarrow O'_{\mathcal{O}_G}(w_1, m_1) \text{ in} \\ \text{let } (w_3, m_3) \leftarrow O'_{\mathcal{O}_F}(w_1, m_2) \text{ in} \end{cases}$$

return $(w_1, m_3)$
\( O'_{O_G} : \lambda x, m. \ (x, w_2) \in L_G, L'_G \rightarrow (w_2, m) \)

\( \top \rightarrow \text{let } w_2 \leftarrow \{0, 1\}^{k_1} \text{ in} \)

\( (w_2, m[L'_G := L'_G :: (x, w_2)]) \)

return \((w_2, m_3)\)

The implementation of \( O_S \) is as before.
Equivalence of PSS and PSS1

Equivalence of PSS and PSS1 is shown using determinization.

- Consider a memory $m$ of PSS1. Let 
  \[ X(m) = (\text{rng} m \cdot L_H) \setminus (\text{dom} m \cdot L'_G) \] and 
  \[ Y(m) = (\text{rng} m \cdot L_H) \setminus (\text{dom} m \cdot L'_F) \] .

- Define $\gamma(m)$ as the uniform distribution over all pairs $(L'_F, L'_G)$ such that $\text{dom} L'_F = X$ and $\text{dom} L'_G = Y$ and typing is respected.

- Then, 
  \[ \text{PSS} \leq_{\text{det}, \gamma} \text{PSS1} \]
Memory: as before + y, $L_u$.
Initialization: y is sampled randomly and $L_u := []$.
$O_{O_H}$ computes $F(w_1)$ and $G(w_1)$ even if they have been already computed.

\[
O_{O_H} : \lambda x, m. (x, w_1) \in L_H \rightarrow \text{return } (w_1, m)
\]

\[
\top \rightarrow \text{let } u \leftarrow \{0, 1\}^k \text{ in}
\]

\[
\text{let } w_1 | w_2 | w_3 = f(u) \otimes y \text{ in}
\]

\[
\text{let } x = x' | r \text{ in}
\]

\[
\text{return } (w_1, m[L_H := L_H :: (x, w_1), \]
\] \[
L'_{F} := L'_{F} :: (w_1, w_3), \]
\] \[
L'_{G} := L'_{G} :: (w_1, w_2 \oplus r) \]
\] \[
L_u := L_u :: (x', r', u, w_1)]
\]
$O_S : \lambda M, m. \quad \text{let } r \leftarrow \{0, 1\}^{k_1} \text{ in}
\quad \text{let } u \leftarrow \{0, 1\}^{k} \text{ in}
\quad \text{let } w_1|w_2|w_3 = f(u) \text{ in}
\quad \text{return } (u, m[L_H := L_H :: (M|r, w_1),}
\quad \quad L_F := L_F :: (w_1, w_3),
\quad \quad L_G := L_G :: (w_1, w_2 \oplus r)])$
Equivalence of PSS1 and PSS2

\[ \text{PSS1} \equiv_{R,\varphi} \text{PSS2} \]

where

\[ R m m' \text{ iff } m \text{ and } m' \text{ agree on } L_H, L_F, L'_F, L_G, L'_G, f, f^{-1} \]

and

\[
\varphi((o, q, a), m_1, m_2) =
\begin{align*}
& o \neq \mathcal{O}_H \land o \neq \mathcal{O}_S \rightarrow \text{return true} \\
& o = \mathcal{O}_H \rightarrow \\
& \text{return } (q \in m_1.L_H) \lor \\
& \quad a \not\in m_1.L_F, m_1.L'_F, m_1.L_G, m_1.L'_G \\
& o = \mathcal{O}_S \rightarrow \\
& \quad \text{let } w_1 = f(a)[k_2] \text{ in} \\
& \quad \text{let } w'_2 = f(a)[k_2 + 1, k_1] \text{ in} \\
& \quad \text{return } w_1 \not\in (m_1.L_F, m_1.L'_F, m_1.L_G, m_1.L'_G) \\
& \quad \land \forall (w_1, g) \in m_2.L_G. (M | w'_2 \oplus w_2) \not\in m_1.L_H
\end{align*}
\]
Assuming the invariant that the length of the list \( L_F, L'_F, L_G, L'_G \), is bounded by \( q_F + q_G + q_H + q_S \),

\[
\Pr[O : \neg \varphi] \leq 2^{-k_2}(q_F + q_G + q_H + q_S) + (q_H + q_S)2^{-k_1}
\]

(UpTo) \[
\begin{array}{c}
\text{PSS1} \equiv \varphi \quad \text{PSS2} \\
\text{(FAIL)} \quad \text{PSS2} : \epsilon_2 \diamond (\neg \varphi) \quad \text{PSS2} : \epsilon_3 \quad \text{EF-CMA}_2 \\
\text{PSS1} : \epsilon_2 + \epsilon_3 \quad \text{EF-CMA}_2
\end{array}
\]

\[
\epsilon_2 = (q_H + q_S)(2^{-k_2}(q_F + q_G + q_H + q_S) + (q_H + q_S)2^{-k_1})
\]
PSS2 : $\epsilon_3 \text{ EF-CMA}_2$ by reduction to the one-way permutation hypothesis.

$$\epsilon_3 = \text{succ}^{ow}(t + (q_H + q_S)T_f + \text{overhead})$$

In summary,

$$\epsilon = \text{succ}^{ow}(t + (q_H + q_S)T_f + \text{overhead}) + \left\{ (q_H + q_S)(2^{-k_2}(q_F + q_G + q_H + q_S) + (q_H + q_S)2^{-k_1}) + 2^{-k_2} \right\}$$
Conclusions

Examples treated with CIL:
- ElGamal, hashed ElGamal.
- FDH, PSS.
- OAEP: IND-CPA, IND-CCA.

Cross fertilization with certicrypt and possibly other tools.
- IND-CCA/OAEP: The insight gained from the proof in CIL enabled the proof in certicrypt (submitted).
- Certicrypt can be used to discharge some of the external reasoning of CIL.

Formalization in a theorem prover:
- Formalization in Coq almost complete (P. Corbineau and help from Ch. Paulin).
- Our main goal concerning this is the soundness of the logic, not necessarily having Coq-certified proofs for cryptosystems.
Further work

- A thorough treatment for the timing aspects (do not care about poly-time, aim at concrete security). Formalization in Coq is under development.

- External reasoning:
  - Reasoning about events: $E \Rightarrow E'$.
  - $\{\text{true}\} O \{ \Pr[\varphi] \leq \epsilon \}$.
  - $0 \equiv_{R, \varphi} 0'$.
  - $0 \leq_{\text{det}, \gamma} 0'$