Computational Indistinguishability Logic*

Gilles Barthe
IMDEA Software
Madrid, Spain

Marion Daubignard
VERIMAG
Grenoble, France

Bruce Kapron
University of Victoria
Canada

Yassine Lakhnech
VERIMAG
Grenoble, France

ABSTRACT

Computational Indistinguishability Logic (CIL) is a logic for reasoning about cryptographic primitives in computational models. It captures reasoning patterns that are common in provable security, such as simulations and reductions. CIL is sound for the standard model, but also supports reasoning in the random oracle and other idealized models. We illustrate the benefits of CIL by formally proving the security of the probabilistic signature scheme (PSS).

1. INTRODUCTION

Cryptography plays a central role in the design of secure and reliable systems. Nevertheless, designing secure cryptographic schemes is notoriously hard. *Provable security* [21] advocates using a mathematical approach to security, in which the security of a cryptographic scheme is formalized as a mathematical statement of the form: "if a security assumption holds for all adversaries, then the security goal is achieved against all adversaries". Moreover, the definition of the adversary, of the security assumption, and of the security goal are themselves subjected to mathematical rigour. Over the years, provable security has become an essential tool for validating the design of cryptographic schemes [36]. Nevertheless, there are concerns that provable security may have reached its limits, and that it must embrace a style of mathematical reasoning that is more amenable to independent verification. In response to these concerns, Halevi [23] advocates building computer-aided verification tools for provable security. Tools like CryptoVerif [8] and CertiCrypt [5] partially fulfill Halevi's suggestion, by providing a rigorous modeling language for describing cryptographic schemes and stating their security, and tool support for checking the correctness of proofs. This approach has the benefit of generality and has been successful in the verification of emblematic examples, e.g. for OAEP encryption and FDH signatures, as well as of a number of protocols.

*A longer version of the paper with proofs is available from `http://software.imdea.org/~gbarthe/CIL/index.html`

A more foundational alternative is to develop models of cryptography that capture at an appropriate level of abstraction the fundamental concepts of provable security. Maurer [29, 28] defines a hierarchy of models and demonstrates how many common concepts can be expressed more crisply by picking the right level of abstraction, and illustrates how this layer of abstractions provides new insights and suggests generalizations of existing concepts. While the potential of this foundational approach is far reaching, it is a challenge to build practical verification tools supporting it. In our view, the main difficulty in reconciling the foundational approach with practical tools lies in the absence of proof systems that capture the common *reasoning principles* that underlie cryptographic proofs. Ideally, one would like to build abstract proof systems that apply across abstraction layers and capture standard cryptographic proof techniques such as reductions or imperfect simulations.

Computational Indistinguishability Logic (CIL) is a logic that supports concise and intuitive proofs across several models of cryptography. Its starting point is the notion of *oracle system*, an abstract model of interactive games in which adaptive adversaries play against a cryptographic scheme by interacting with oracles. Oracle systems are inspired by probabilistic process algebra, but do not commit to a particular model or syntax. As a result, they provide a unified foundation for cryptographic games, can be formalized neatly in a proof assistant, and capture both the standard model and idealized models such as the random oracle model or ideal ciphertext model. Moreover, oracle systems provide a unifying semantics for the different languages used in practical tools for cryptographic proofs: mathematics [33], processes [8], λ-calculus [3], imperative programs [5].

CIL features a small set of rules that capture common reasoning patterns, e.g. simulations and reductions steps. The soundness of these rules may be established using (mild variants of) concepts that are well understood by the programming language and concurrency communities, such as contexts and bisimulations. Moreover, CIL features interface rules to connect with external reasoning. The use of external reasoning offers a number of advantages. First of all, it enforces a separation between proof steps which are purely logical or information-theoretic from those which directly involve (reduction-based) security. Secondly, it allows a presentation of the proof system which does not commit to a particular syntactic representation for oracle systems – rules specific to a particular representation may be introduced as "plug-ins" to the rules as presented. Thus, rules for establishing external premises are not considered as part of

CIL. Instead, existing proof techniques, including automated techniques, are used to establish external premises.

Although our long term objective is to enhance practical tools for verifying cryptographic schemes, this paper focuses on theoretical foundations of CIL. To illustrate the applicability of CIL, we consider the Probabilistic Signing Scheme (PSS), a widely used signature scheme that forms part of the PKCS standard [6]; however, we have also used CIL to prove exact security of several other signatures and encryption primitives, including EF-CMA of FDH and IND-CCA2 of OAEP.

Contents. The main technical contributions of the paper are: i) an abstract framework to capture cryptographic games as oracle systems (Section 2); ii) reasoning tools for oracle systems: context application (Section 4), bisimulations and determinization (Section 5); iii) a formal proof in CIL of PSS (Section 8).

Preliminaries. We use standard notation, e.g. $\mathbf{1}$ to denote the unit type, (x, y) to denote pairs, and match ... with notation for pattern-matching. When pattern-matching is driven by types, as for splitting a bitstring of size k into bitstrings of size k_0 and k_1 with $k = k_0 + k_1$, we write

$$\mathsf{match}\ x_0 : \{0,1\}^{k_0} \mid x_1 : \{0,1\}^{k_1}\ \mathsf{with}\ y\ \mathsf{in}$$

For a set A, $\mathcal{D}(A)$ denotes the set of distributions over A. For a distribution with probability function p, we have an associated random variable X, and for $a \in A$ we write $\Pr(X = a)$ for $p(a)$. For $a \in A$, $\delta(a)$ denotes the Dirac distribution on A. For the sake of readability, we often describe distributions in a style that is closer to programming than to standard mathematics. We use monadic operators for subdistributions, and add the subdistributions produced by each return statement to obtain a distribution. Unit operators that map a value to a distribution are sometimes ommitted. We use _ to denote arguments that are not used, or elements of tuples whose value is irrelevant in the final distribution. Finally, a distribution d over A lies in the range of a predicate P over A, written $\mathsf{range}\ d\ A$ iff $P\ a$ for every $a \in A$ such that $d\ a > 0$.

2. ORACLE SYSTEMS

Our starting point is a general framework for modeling the interaction between an adversary and a cryptographic scheme with oracles.

2.1 Oracle systems and adversaries

An oracle system is a stateful system that provides oracle access to adversaries.

Definition 1. *An* oracle system $\mathbb{O}$ *is given by:*

- *sets* M_o *of oracle memories and* $\mathsf{N_o}$ *of oracles;*
- *for each* $o \in \mathsf{N_o}$*, a query domain* $\mathsf{In}(o)$*, an answer domain* $\mathsf{Out}(o)$ *and an implementation:*

$$\mathsf{O}_o : \mathsf{In}(o) \times \mathsf{M}_o \to \mathcal{D}(\mathsf{Out}(o) \times \mathsf{M}_o)$$

- *a distinguished initial memory* $\bar{m}_o \in \mathsf{M}_o$*, and distinguished oracles* o_{I} *for initialization and* o_{F} *for finalization, such that* $\mathsf{In}(o_{\mathrm{I}}) = \mathsf{Out}(o_{\mathrm{F}}) = \mathbf{1}$*. We let* $\mathsf{Res} = \mathsf{In}(o_{\mathrm{F}})$*.*

Two oracle systems $\mathbb{O}$ and $\mathbb{O}'$ are compatible iff they have the same sets of oracle names, and the query and answer domains of each oracle name coincide in both oracle systems. When building a compatible oracle system from another one, it is thus sufficient to provide its set of memories, its initial memory and the implementation of its oracles.

Adversaries interact with oracle systems by making queries, and receiving answers. An exchange (for an oracle system $\mathbb{O}$) is a triple (o, q, a) where $o \in \mathsf{N_o}$, $q \in \mathsf{In}(o)$ and $a \in \mathsf{Out}(o)$; we let Xch be the set of exchanges. Initial and final exchanges are defined in the obvious way, by requiring that o is an initialization and finalization oracle resp; the sets of initial and final exchanges are denoted by $\mathsf{Xch}_{\mathrm{I}}$ and $\mathsf{Xch}_{\mathrm{F}}$ respectively. The sets Que of queries and Ans of answers are respectively defined as $\{(o, q) \mid (o, q, a) \in \mathsf{Xch}\}$ and $\{(o, a) \mid (o, q, a) \in \mathsf{Xch}\}$.

Definition 2. *An* adversary $\mathbb{A}$ *(for an oracle system* $\mathbb{O}$*) is given by a set* M_a *of adversary memories, an initial memory* $\bar{m}_a \in \mathsf{M}_a$ *and functions for querying, and updating:*

$$\begin{array}{lcl} \mathsf{A} & : & \mathsf{M}_a \to \mathcal{D}(\mathsf{Que} \times \mathsf{M}_a) \\ \mathsf{A}_{\downarrow} & : & \mathsf{Xch} \times \mathsf{M}_a \to \mathcal{D}(\mathsf{M}_a) \end{array}$$

Informally, the interaction between an oracle system and an adversary proceeds in three successive phases: the initialization oracle sets the initial memory distributions of the oracle system and of the adversary. Then, $\mathbb{A}$ performs computations, updates its state and submits queries to $\mathbb{O}$. In turn, $\mathbb{O}$ performs computations, updates its state, and replies to $\mathbb{A}$, which updates its state. Finally, $\mathbb{A}$ outputs a result by calling the finalization oracle.

2.2 Semantics

The purpose of this section is to define formally the interaction between oracle systems and adversaries, using (probabilistic) transition systems.

Definition 3. *A* transition system $\mathcal{S}$ *consists of:*

- *a (countable non-empty) set* M *of memories (states), with a distinguished initial memory* $\bar{m}$*;*
- *a set* Σ *of actions, with distinguished subsets of* Σ_{I} *and* Σ_{F} *of initialization and finalization actions;*
- *a (partial) transition function* $\mathsf{step} : \mathsf{M} \rightharpoonup \mathcal{D}(\Sigma \times \mathsf{M})$*.*

A partial execution sequence of $\mathcal{S}$ is a sequence η of the form $m_0 \xrightarrow{x_1} m_1 \xrightarrow{x_2} \ldots \xrightarrow{x_k} m_k$ such that

$$\Pr[\mathsf{step}(m_{k-1}) = (a_k, m_k)] > 0$$

for $i = 1 \ldots k$ and $x_i = (o_i, q_i, a_i)$. If $k = 1$, then η is a step. If $m_0 = \bar{m}$, and $x_1 \in \Sigma_{\mathrm{I}}$ and $x_k \in \Sigma_{\mathrm{F}}$, then η is an execution sequence of length k. A probabilistic transition system $\mathcal{S}$ induces a sub-distribution on executions, denoted $\mathcal{S}$, s.t. the probability of a finite execution sequence η is

$$\Pr[\mathcal{S} = \eta] = \Pi_{i=1}^{k} \Pr[\mathsf{step}(m_{i-1}) = (a_i, m_i)]$$

A transition system is of height $k \in \mathbb{N}$ if all its executions have length at most k; in this case, $\mathcal{S}$ is a distribution.

Definition 4. *Let* $\mathbb{O}$ *be an oracle system and* $\mathbb{A}$ *be an* $\mathbb{O}$*-adversary . The composition* $\mathbb{A} \mid \mathbb{O}$ *is a transition system such that* $\mathsf{M} = \mathsf{M}_a \times \mathsf{M}_o$*, the initial memory is* $(\bar{m}_a, \bar{m}_o)$*, the*

set of actions is $\Sigma = \mathsf{Xch}$*, and* $\Sigma_{\mathrm{I}} = \mathsf{Xch}_{\mathrm{I}}$ *and* $\Sigma_{\mathrm{F}} = \mathsf{Xch}_{\mathrm{F}}$*, and*

$$\begin{array}{rcl}\mathsf{step}_{\mathbb{A}|\mathbb{O}}(m_a, m_o) & \stackrel{def}{=} & \mathsf{let}\ ((o,q),m_a') \leftarrow \mathsf{A}(m_a)\ \mathsf{in} \\ & & \mathsf{let}\ (a,m_o') \leftarrow \mathsf{O}_o(q,m_o)\ \mathsf{in} \\ & & \mathsf{let}\ m_a'' \leftarrow \mathsf{A}_{\downarrow}((o,q,a),m_a')\ \mathsf{in} \\ & & \mathsf{return}\ ((o,q,a),(m_a'',m_o'))\end{array}$$

An adversary is called k-bounded, if $\mathbb{A} \mid \mathbb{O}$ is of height k. This means that $\mathbb{A}$ calls the finalization oracle after less than k interactions with $\mathbb{O}$. (Note that $\mathbb{A} \mid \mathbb{O}$ may be ill-defined for unbounded adversaries, since $\mathsf{step}_{\mathbb{A}|\mathbb{O}}(m_a, m_o)$ may be a sub-distribution.) Throughout the paper, we only consider bounded adversaries, i.e. that are k-bounded for some k.

2.3 Events

Security properties abstract away from the state of adversaries, and are modeled using traces. Informally, a trace τ is an execution sequence η from which the adversary memories have been erased.

Definition 5. *Let* $\mathbb{O}$ *be an oracle system.*

- *A* partial trace *is a sequence* τ *of the form*

$$m_0 \xrightarrow{x_1} m_1 \xrightarrow{x_2} \ldots \xrightarrow{x_k} m_k$$

where $m_0 \ldots m_k \in \mathsf{M}_o$ *and* $x_1 \ldots x_k \in \mathsf{Xch}$ *such that*

$$\Pr[\mathsf{O}_{o_i}(q_i, m_{i-1}) = (a_i, m_i)] > 0$$

for $i = 1 \ldots k$ *and* $x_i = (o_i, q_i, a_i)$*. A* trace *is a partial trace* τ *such that* $m_0 = \bar{m}_o$*, and* $x_1 = (o_{\mathrm{I}}, \_, \_)$ *and* $x_k = (o_{\mathrm{F}}, \_, \_)$*.*

- *An* $\mathbb{O}$*-event* E *is a predicate over* $\mathbb{O}$*-traces, whereas an extended* $\mathbb{O}$*-event* E *is a predicate over partial* $\mathbb{O}$*-traces.*

The probability of an (extended) event is derived directly from the definition of $\mathbb{A} \mid \mathbb{O}$: since each execution sequence η induces a trace $\mathcal{T}(\eta)$ simply by erasing the adversary memory at each step, one can define for each trace τ the set $\mathcal{T}^{-1}(\tau)$ of execution sequences that are erased to τ, and for every (generalized) event E the probability:

$$\begin{array}{rcl}\Pr(\mathbb{A}|\mathbb{O} : E) & = & \Pr(\mathbb{A}|\mathbb{O} : \mathcal{T}^{-1}(E)) \\ & = & \sum_{\{\eta \in \mathsf{Exec}(\mathbb{A}|\mathbb{O}) \mid E(\mathcal{T}(\eta)) = \mathsf{true}\}} \Pr(\mathbb{A}|\mathbb{O} : \eta)\end{array}$$

Constructions and proofs in CIL use several common operations on (extended) events and traces. First, one can define the conjunction, disjunction, etc of events; moreover, one can define for every predicate P over $\mathsf{Xch} \times \mathsf{M}_o \times \mathsf{M}_o$ the events "eventually P" F_P and "always P" G_P that correspond to P being satisfied by one step and all steps of the trace respectively. Moreover, let E be an (extended) event; then we can define the event "E until P" $E\mathsf{U}P$ as follows. For a trace τ of the form:

$$m_0 \xrightarrow{x_1} m_1 \xrightarrow{x_2} \ldots \xrightarrow{x_k} m_k$$

we set $(E\mathsf{U}\neg\varphi)(\tau) = \mathsf{true}$ iff there is a minimal $i \in [1,k]$ such that $(x_i, m_{i-1}, m_i) \notin \varphi$ and $E(\tau[i-1]) = \mathsf{true}$, where $\tau[i-1]$ is the partial trace:

$$m_0 \xrightarrow{x_1} m_1 \xrightarrow{x_2} \ldots \xrightarrow{x_{i-1}} m_{i-1}$$

Intuitively, $E\mathsf{U}\neg\varphi$ holds when $\neg\varphi$ holds at some transition in the trace and E holds before that. This type of temporal reasoning turns out to be useful in proofs where the order of events is important.

Reduction-based arguments require that adversaries can partially simulate behaviors. In some cases, adversaries must test whether a predicate $\varphi \subseteq \mathsf{Xch} \times \mathsf{M}_o \times \mathsf{M}_o$ holds for given values. Since the adversary has no access to the oracle memory, we say that φ is testable iff for all x, m_1, m_1', m_2, m_2', $\varphi(x, m_1, m_1')$ iff $\varphi(x, m_2, m_2')$.

We now turn to traces. Given two traces τ and τ', we write $\tau \ R \ \tau'$ iff for every $i = 1 \ldots k$, we have $m_i \ R \ m_i'$, where:

$$\begin{array}{rcl}\tau & = & m_0 \xrightarrow{x_1} m_1 \xrightarrow{x_2} \ldots \xrightarrow{x_k} m_k \\ \tau' & = & m_0' \xrightarrow{x_1} m_1' \xrightarrow{x_2} \ldots \xrightarrow{x_k} m_k'\end{array}$$

Moreover, we say that two events E and E' are R-compatible, written $E \ R \ E'$, iff $E(\tau)$ is equivalent to $E'(\tau')$ for every traces τ and τ' such that $\tau \ R \ \tau'$.

3. CIL: STATEMENTS AND BASIC RULES

This introduces the judgments and basic rules of CIL. Subsequent sections provide additional rules that are used to carry reduction arguments (Section 4), simulation arguments (Section 5) and interprocedural code motion, e.g. eager sampling (Section 6).

3.1 Judgments

CIL considers negligibility statements of the form $\mathbb{O} :_{\epsilon} E$, where E is an event. A statement $\mathbb{O} :_{\epsilon} E$ is *valid*, written $\models \mathbb{O} :_{\epsilon} E$, iff for every adversary $\mathbb{A}$,

$$\Pr(\mathbb{A}|\mathbb{O} : E) \leq \epsilon$$

We also consider indistinguishability statements of the form $\mathbb{O} \sim_{\epsilon} \mathbb{O}'$, where $\mathbb{O}$ and $\mathbb{O}'$ are compatible oracle systems which expect a boolean as result. A statement $\mathbb{O} \sim_{\epsilon} \mathbb{O}'$ is *valid*, written $\models \mathbb{O} \sim_{\epsilon} \mathbb{O}'$, iff for every adversary $\mathbb{A}$,

$$|\Pr(\mathbb{A}|\mathbb{O} : \mathrm{R} = \mathsf{true}) - \Pr(\mathbb{A}|\mathbb{O}' : \mathrm{R} = \mathsf{true})| \leq \epsilon$$

where $\mathrm{R} = \mathsf{true}$ is shorthand for $\mathsf{F}_{\lambda(o,q,r).o=o_{\mathrm{F}} \wedge (r=\mathsf{true})}$. CIL statements support faithful definitions of standard security assumptions such as DDH, or one-way permutations, and security properties such as IND-CPA, IND-CCA, or EF-CMA.

As cryptographic proofs often rely on assumptions, CIL manipulates sequents of the form $\Delta \Longrightarrow \phi$, where Δ is a set of statements (the assumptions), and ϕ is a statement (the conclusion). Validity extends to sequents $\Delta \Longrightarrow \phi$ in the usual manner. Given a set Δ of statements, $\models \Delta$ iff $\models \psi$ for every $\psi \in \Delta$. Then $\Delta \models \phi$ iff $\models \Delta$ implies $\models \phi$. For clarity and brevity, our presentation of CIL omits hypotheses, and the standard structural and logical rules for sequent calculi.

3.2 Basic rules

The first set of rules supports equational reasoning:

$$\frac{}{\mathbb{O} \sim_0 \mathbb{O}} \qquad \frac{\mathbb{O} \sim_{\epsilon} \mathbb{O}'}{\mathbb{O}' \sim_{\epsilon} \mathbb{O}} \qquad \frac{\mathbb{O} \sim_{\epsilon} \mathbb{O}' \qquad \mathbb{O}' \sim_{\epsilon'} \mathbb{O}''}{\mathbb{O} \sim_{\epsilon+\epsilon'} \mathbb{O}''}$$

CIL features a rule that bears some similarity with the rule of consequence in Hoare logic, has many useful instances, and is trivially valid:

$$\frac{\mathbb{O} :_{\epsilon_i} E_i \ (i \in I) \qquad E \Rightarrow \bigvee_{i \in I} E_i}{\mathbb{O} :_{\sum_{i \in I} \epsilon_i} E}\ \mathrm{UR}$$

CIL also features the trivially sound rule:

$$\frac{\mathbb{O}\{E\}}{\mathbb{O} :_0 E}\ \text{POST-S}$$

where the statement is used $\mathbb{O}\{E\}$ to indicate that an event E holds for every execution of $\mathbb{A}|\mathbb{O}$, for all adversaries $\mathbb{A}$. Statements of the form $\mathbb{O}\{E\}$ can be established using Hoare logics for probabilistic programs.

Besides, CIL features a rule to compute an upper bound on the probability of an event from the number of oracle calls, and from the probability that a single oracle call triggers that event. Let φ be a predicate on $\mathsf{Xch} \times \mathsf{M}_o \times \mathsf{M}_o$, and define for every $o \in \mathsf{N_o}$ the probability ϵ_o as

$$\max_{\substack{q\in\mathsf{Que},m\in\mathsf{M}_o\\ a\in\mathsf{Ans},m'\in\mathsf{M}_o}} \Pr[\mathsf{O}_o(q,m) = (a,m') \wedge \varphi((o,q,a),m,m')]$$

For every $o \in \mathsf{N_o}$ let k_o denote the maximal number of queries to o, and let $\epsilon = \sum_{o\in\mathsf{N_o}} k_o\epsilon_o$. CIL features the rule:

$$\frac{}{\mathbb{O} :_\epsilon \mathsf{F}_\varphi}\ \text{FAIL}$$

This rule is used repeatedly in the proof of PSS, and is used in many other examples, such as FDH or OAEP. More general rules are sometimes required, e.g. for the Switching Lemma. These rules are ommitted.

3.3 A remark on the adversarial model

CIL judgments often make implicit assumptions on the number of oracle calls permitted to the adversary. Typically, a CIL statement will be of the form $\mathbb{O} :_\epsilon E$, where ϵ is an expression that depends on the maximal number of calls that an adversary is allowed to make to any given oracle. In other words, ϵ is implicitly understood as a function of type

$$(\mathsf{N_o} \to \mathbb{N}) \to [0,1]$$

More generally, it is possible to view ϵ as a function of type

$$((\mathsf{N_o} \to \mathbb{N}) \times \mathbb{T}) \to [0,1]$$

where $\mathbb{T}$ denotes the execution time of the adversary.

Alternatively, one often considers restricted classes of adversaries, e.g. adversaries that execute in probabilistic polynomial time. Our results can be modified to accomodate classes of adversaries; this is achieved by postulating closure properties of these classes, and by ensuring a correct usage of these security hypotheses. For example, preservation of indistinguishability under contexts, as stated in Section 4 becomes: for every class Adv of adversaries that is closed with respect to $\mathbb{C} \mid \cdot$, if $\mathbb{O} \sim \mathbb{O}'$ for every adversary $\mathbb{A} \in \mathsf{Adv}$, then $\mathbb{C}[\mathbb{O}] \sim \mathbb{C}[\mathbb{O}]'$ for every adversary $\mathbb{A} \in \mathsf{Adv}$. Note that making the adversarial model parametric is viewed as an important benefit of abstraction in [28].

4. CONTEXTS

This section introduces contexts, which provide a main tool to perform reduction arguments. Informally, a context $\mathbb{C}$ is an intermediary between an oracle system $\mathbb{O}$ and adversaries. One can compose a $\mathbb{O}$-context $\mathbb{C}$ with $\mathbb{O}$ to obtain a new oracle system $\mathbb{C}[\mathbb{O}]$ and with a $\mathbb{C}[\mathbb{O}]$-adversary to obtain a new $\mathbb{O}$-adversary $\mathbb{C} \parallel \mathbb{A}$. Moreover, one can show that the systems $\mathbb{C} \parallel \mathbb{A} \mid \mathbb{O}$ and $\mathbb{A} \mid \mathbb{C}[\mathbb{O}]$ coincide in a precise mathematical sense. Despite its seemingly naivety, the relationship captures many reduction arguments used in cryptographic proofs and yields CIL rules that allow proving many schemes.

The definition of contexts is very similar to that of oracle system, except that procedures are implemented by two functions, one that transfers calls from the adversary to the oracles, and another one that transfers answers from the oracles to the adversary—possibly after some computations.

DEFINITION 6. *An $\mathbb{O}$-context $\mathbb{C}$ is given by:*

- *sets M_c of context memories, an initial memory $\bar{m}_c$ and $\mathsf{N_c}$ of procedures;*
- *for every $c \in \mathsf{N_c}$, a query domain $\mathsf{In}(c)$, an answer domain $\mathsf{Out}(c)$, and two functions:*

$$\mathsf{C}_c^{\rightarrow} : \mathsf{In}(c) \times \mathsf{M}_c \to \mathcal{D}(\mathsf{Que} \times \mathsf{M}_c)$$
$$\mathsf{C}_c^{\leftarrow} : \mathsf{In}(c) \times \mathsf{Xch} \times \mathsf{M}_c \to \mathcal{D}(\mathsf{Out}(c) \times \mathsf{M}_c)$$

- *distinguished initial and finalization procedures c_{I} and c_{F} s.t. $\mathsf{In}(c_{\mathrm{I}}) = \mathsf{Out}(c_{\mathrm{F}}) = \mathbf{1}$, and for all x and m_c:*

$$\mathsf{range}\ (\mathsf{C}_{c_{\mathrm{I}}}^{\rightarrow}(x,m_c))\ (\lambda((o,\_),\_).\ o = o_{\mathrm{I}})$$
$$\mathsf{range}\ (\mathsf{C}_{c_{\mathrm{F}}}^{\rightarrow}(x,m_c))\ (\lambda((o,\_),\_).\ o = o_{\mathrm{F}})$$

We let $\mathsf{Res}_c = \mathsf{In}(c_{\mathrm{F}})$.

An indistinguishability context *is a $\mathbb{O}$-context $\mathbb{C}$ such that $\mathsf{Res}_c = \mathsf{Res}$ and $\mathsf{C}_{c_{\mathrm{F}}}^{\rightarrow}(r,m) = \delta_{((r,o_{\mathrm{F}}),m)}$ for all r and m.*

The sets Que_c of context queries, Ans_c of context answers, and Xch_c of context exchanges are defined similarly to oracle systems.

An $\mathbb{O}$-context can be composed with the oracle system $\mathbb{O}$ or with any $\mathbb{O}$-adversary $\mathbb{A}$, yielding a new oracle system $\mathbb{C}[\mathbb{O}]$ or a new adversary $\mathbb{C} \parallel \mathbb{A}$. We begin by defining the composition of a context and an oracle system.

DEFINITION 7. *The application of an $\mathbb{O}$-context $\mathbb{C}$ to $\mathbb{O}$ defines an oracle system $\mathbb{C}[\mathbb{O}]$ such that:*

- *the set of memories is $\mathsf{M}_c \times \mathsf{M}_o$, 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:*

$$\begin{array}{l}\lambda q_c, (m_c, m_o).\\ \quad \mathsf{let}\ ((o,q_o),m_c') \leftarrow \mathsf{C}_c^{\rightarrow}(q_c,m_c)\ \mathsf{in}\\ \quad \mathsf{let}\ (a_o,m_o') \leftarrow \mathsf{O}_o(q_o,m_o)\ \mathsf{in}\\ \quad \mathsf{let}\ (a_c,m_c'') \leftarrow \mathsf{C}_c^{\leftarrow}(q_c,(o,q_o,a_o),m_c')\ \mathsf{in}\\ \quad \mathsf{return}\ (a_c,(m_c'',m_o'))\end{array}$$

where the $\mathsf{let}\ \cdot \leftarrow \cdot\ \mathsf{in}$ notation is used for monadic composition, and return is used for returning the result of the function.

The composition of an adversary with a context is slightly more subtle and requires that the new adversary stores the current query in its state.

DEFINITION 8. *The application of an $\mathbb{O}$-context $\mathbb{C}$ to a $\mathbb{C}[\mathbb{O}]$ adversary $\mathbb{A}$ defines a $\mathbb{O}$-adversary $\mathbb{C} \parallel \mathbb{A}$ such that:*

- *the set of memories is $\mathsf{M}_c \times \mathsf{M}_a \times \mathsf{Que}_c$, and the initial memory is $(\bar{m}_c, \bar{m}_a, \_)$;*

- *the transition function is:*

$$\begin{array}{l}\lambda(m_c, m_a, \_).\\ \quad \mathsf{let}\ ((c, q_c), m'_a) \leftarrow \mathsf{A}(m_a)\ \mathsf{in}\\ \quad \mathsf{let}\ ((o, q), m'_c) \leftarrow \mathsf{C}_c^{\rightarrow}(q_c, m_c)\ \mathsf{in}\\ \quad \mathsf{return}\ ((o, q), (m'_c, m'_a, (o, q)))\end{array}$$

- *the update function is:*

$$\begin{array}{l}\lambda((m_c, m_a, (o_c, q_c)), (o_o, q_o, a_o)).\\ \quad \mathsf{let}\ (a_c, m'_c) \leftarrow \mathsf{C}_c^{\leftarrow}(q_c, (o_o, q_o, a_o), m_c)\ \mathsf{in}\\ \quad \mathsf{return}\ (m'_c, \mathsf{A}_{\downarrow}((o_c, q_c, a_c), m_a), \_)\end{array}$$

Likewise, one defines the $\mathbb{O}$-event $E \circ \mathbb{C}$ as the composition of the $\mathbb{C}[\mathbb{O}]$-event E with the context $\mathbb{C}$. The composition relies on defining mixed $\mathbb{C}[\mathbb{O}]$-traces with steps of the form

$$(m_c, m_o) \xrightarrow{(x,y)} (m''_c, m'_o)$$

where $x = (o, q_o, a_o)$ and $Y = (c, q_c, a_c)$ are defined according to Definition 7 and there exists m'_C such that

$$\begin{array}{rcl}\Pr[\mathsf{C}_c^{\rightarrow}(q_c, m_c) = ((o, q_o), m'_c)] & > & 0\\ \Pr[\mathsf{O}_o(q_o, m_o) = (a_o, m'_o)] & > & 0\\ \Pr[\mathsf{C}_c^{\leftarrow}(q_c, (o, q_o, a_o), m'_c)] = (a_c, m''_c)] & > & 0\end{array}$$

Mixed traces can be projected to $\mathbb{C}[\mathbb{O}]$-traces and to $\mathbb{O}$-traces; we denote $\pi_{\mathbb{C}}[\mathbb{O}]$ and $\pi_{\mathbb{O}}$ the projections to $\mathbb{C}[\mathbb{O}]$-traces and $\mathbb{O}$-traces respectively. Then, each $\mathbb{C}[\mathbb{O}]$-event E yields a predicate E_{mix} over mixed traces, and then a predicate over $\mathbb{O}$-traces, defined respectively as:

$$\begin{array}{l}\lambda\tau_{\text{mix}}.\ E(\pi_{\mathbb{C}}[\mathbb{O}](\tau))\\ \lambda\tau.\ \forall\tau_{\text{mix}}.\ \pi_{\mathbb{O}}(\tau_{\text{mix}}) = \tau \Rightarrow E_{\text{mix}}(\tau_{\text{mix}})\end{array}$$

Proposition 1. *Let $\mathbb{O}, \mathbb{O}'$ be compatible oracle systems and $\mathbb{C}$ be an $\mathbb{O}$-context.*

- *If $\mathbb{C}$ is an indistinguishability context and $\models \mathbb{O} \sim_\epsilon \mathbb{O}'$ then $\models \mathbb{C}[\mathbb{O}] \sim_\epsilon \mathbb{C}[\mathbb{O}']$.*
- *For all $\mathbb{C}[\mathbb{O}]$-event E, if $\models \mathbb{O} :_\epsilon E \circ \mathbb{C}$ then $\models \mathbb{C}[\mathbb{O}] :_\epsilon E$.*

5. BISIMULATION

Game-based proofs often proceed by transforming an oracle system into an equivalent one, or in case of imperfect simulation into a system that is equivalent up to some bad event. This section justifies this reasoning in terms of probabilistic transition systems, using a mild extension of the standard notion of bisimulation. More specifically, we define the notion of *bisimulation up to*, where two probabilistic transition systems are bisimilar until the failure of a condition on their transitions. The definition of bisimulation is recovered by considering bisimulations up to the constant predicate true.

Let $\mathbb{O}$ and $\mathbb{O}'$ be two compatible oracle systems. For every oracle name, we let $\hat{\mathsf{M}}$ be $\mathsf{M}_o + \mathsf{M}'_o$ and for every $o \in \mathsf{N_o}$, we let $\hat{\mathsf{O}}_o$ be the disjoint sum of O_o and O'_o, i.e.

$$\hat{\mathsf{O}}_o : \mathsf{In}(o) \times \hat{\mathsf{M}} \to \mathcal{D}(\mathsf{Out}(o) \times \hat{\mathsf{M}})$$

We write $m \xrightarrow{(o,q,a)}_{>0} m'$ iff $\Pr[\hat{\mathsf{O}}_o(q, m_i) = (a, m'_i)] > 0$.

Definition 9. *Let $\varphi \subseteq \mathsf{Xch} \times \hat{\mathsf{M}} \times \hat{\mathsf{M}}$ and let $R \subseteq \hat{\mathsf{M}} \times \hat{\mathsf{M}}$ be an equivalence relation. $\mathbb{O}$ and $\mathbb{O}'$ are bisimilar up to φ, written $\mathbb{O} \equiv_{R,\varphi} \mathbb{O}'$, iff $\bar{m}\ R\ \bar{m}'$, and for all $m_1 \xrightarrow{(o,q,a)}_{>0} m'_1$ and $m_2 \xrightarrow{(o,q,a)}_{>0} m'_2$ such that $m_1\, R\, m_2$:*

- stability: *if $m'_1\, R\, m'_2$ then*

$$\varphi((o, q, a), m_1, m'_1) \quad \Leftrightarrow \quad \varphi((o, q, a), m_2, m'_2)$$

- compatibility: *if $\varphi((o, q, a), m_1, m'_1)$, then*

$$\Pr[\hat{\mathsf{O}}_o(q, m_1) \in (a, C)] = \Pr[\hat{\mathsf{O}}_o(q, m_2) \in (a, C)]$$

where C is the equivalence class of m'_1 under R.

Bisimulations are closely related to observational equivalence and relational Hoare logic, and allow to justify proofs by simulations. Besides, bisimulations up to subsume the Fundamental Lemma of [35].

Proposition 2. *For all compatible oracle systems $\mathbb{O}$ and $\mathbb{O}'$, every relation R and predicate φ s.t. $\mathbb{O} \equiv_{R,\varphi} \mathbb{O}'$, every R-compatible pair of events E and E', and adversary $\mathbb{A}$:*

- $\Pr(\mathbb{A} \mid \mathbb{O} : E \wedge \mathsf{G}_\varphi) = \Pr(\mathbb{A} \mid \mathbb{O}' : E' \wedge \mathsf{G}_\varphi)$
- $\Pr(\mathbb{A} \mid \mathbb{O} : \mathsf{F}_{\neg\varphi}) = \Pr(\mathbb{A} \mid \mathbb{O} : \mathsf{F}_{\neg\varphi})$

If moreover φ is testable and E and E' are R-compatible extended events, then there exists an adversary $\mathbb{A}^\varphi$ s.t.

$$\Pr(\mathbb{A} \mid \mathbb{O} : E\mathsf{U}\neg\varphi) = \Pr(\mathbb{A}^\varphi \mid \mathbb{O}' : E'\mathsf{U}\neg\varphi)$$

6. DETERMINIZATION

Bisimulation is stronger than language equivalence, and cannot always be used to hop from one game to another. In particular, bisimulation cannot be used for eager/lazy sampling, or for extending the internal state of the oracle system. The goal of this section is to introduce a general construction, inspired from the subset construction for determinizing automata, to justify such transitions.

Definition 10. *Let $\mathbb{O}$ and $\mathbb{O}'$ be compatible oracle systems.* $\mathbb{O}$ determinizes $\mathbb{O}'$ *by $\gamma : \mathsf{M}_o \to \mathcal{D}(\mathsf{M}''_o)$, written $\mathbb{O} \leq_{\text{det},\gamma} \mathbb{O}'$, iff $\mathsf{M}_o \times \mathsf{M}''_o = \mathsf{M}'_o$, and there exists $\bar{m}''_o$ such that $(\bar{m}_o, \bar{m}''_o) = \bar{m}'_o$, and $\gamma(\bar{m}_o) = \delta_{\bar{m}''_o}$, and:*

$$\Pr[\gamma(m_2) = m'_2]\ p_1 = \sum_{m'_1 \in \mathsf{M}''_o} \Pr[\gamma(m_1) = m'_1]\ p_2(m'_1)$$

for all $m, m_1, m_2 \in \mathsf{M}_o$, $m', m'_2 \in \mathsf{M}''_o$, where:

$$\begin{array}{rcl}p_1 & = & \Pr[\mathsf{O}(o_c, q, m_1) = (a, m_2)]\\ p_2(m'_1) & = & \Pr[\mathsf{O}'(o_c, q, (m_1, m'_1)) = (a, (m_2, m'_2))]\end{array}$$

We define a projection function π from $\mathbb{O}'$-traces to $\mathbb{O}$-traces by extending the projection from $\mathsf{M}_o \times \mathsf{M}''_o$ to M_o to traces.

Proposition 3. *Let $\mathbb{O}$ and $\mathbb{O}'$ be such that $\mathbb{O} \leq_\gamma \mathbb{O}'$, and let E be a $\mathbb{O}$-event. For every $\mathbb{O}$-adversary $\mathbb{A}$:*

$$\Pr(\mathbb{A}|\mathbb{O} : E) = \Pr(\mathbb{A}|\mathbb{O}' : E \circ \pi)$$

We conclude this section by observing that is possible to combine determinization and bisimulation in a single concept. By doing so, one obtains stronger proof rules that yield more compact proofs. To simplify the presentation, we chose to keep the two notions separate.

7. CIL: RULES AND SOUNDNESS

This section introduces additional rules that can be derived from the results of the preceding sections, and states the soundness of the logic. It also discusses methods to establish external premisses that are used in CIL rules.

7.1 Rules for contexts and oracles

First, CIL features composition rules that are an immediate application of the results of Section 4:

$$\frac{\mathbb{O} \sim_\epsilon \mathbb{O}'}{\mathbb{C}[\mathbb{O}] \sim_\epsilon \mathbb{C}[\mathbb{O}']}\ \text{SUB} \qquad \frac{\mathbb{O} :_\epsilon E \circ \mathbb{C}}{\mathbb{C}[\mathbb{O}] :_\epsilon E}\ \text{NegSUB}$$

Then, CIL feature rules for oracles. These rules are consequences of the results of Section 5, and involve equality of oracle systems (up to φ). The rules (NegOR$\forall$), (Neg$\Diamond$) and (OR) are consequences of Proposition 2:

$$\frac{\mathbb{O} :_\epsilon E \wedge \mathsf{G}_\varphi \quad \mathbb{O} \equiv_{R,\varphi} \mathbb{O}' \quad E\ R\ E'}{\mathbb{O}' :_\epsilon E' \wedge \mathsf{G}_\varphi}\ \text{NegOR}\forall$$

$$\frac{\mathbb{O} :_\epsilon \mathsf{F}_{\neg\varphi} \quad \mathbb{O} \equiv_{R,\varphi} \mathbb{O}'}{\mathbb{O}' :_\epsilon \mathsf{F}_{\neg\varphi}}\ \text{Neg}\Diamond$$

$$\frac{\mathbb{O} :_\epsilon \mathsf{F}_{\neg\varphi} \quad \mathbb{O} \equiv_{R,\varphi} \mathbb{O}'}{\mathbb{O} \sim_\epsilon \mathbb{O}'}\ \text{OR}$$

The rules (NegDET) and its counterpart rule (DET) are consequences of Proposition 4:

$$\frac{\mathbb{O} \leq_\gamma \mathbb{O}' \quad \mathbb{O}' :_\epsilon E \circ \pi}{\mathbb{O} :_\epsilon E}\ \text{NegDET}$$

$$\frac{\mathbb{O} \leq_\gamma \mathbb{O}' \quad \mathbb{O} :_\epsilon E}{\mathbb{O}' :_\epsilon E \circ \pi}\ \text{NegDET}$$

$$\frac{\mathbb{O} \leq_\gamma \mathbb{O}'}{\mathbb{O} \sim_0 \mathbb{O}'}\ \text{DET}$$

The rule (NegOR$\exists$) captures imperfect simulations. It is a consequence of the Proposition 2. In this rule, E and E' are extended events:

$$\frac{\mathbb{O} :_\epsilon E\mathsf{U}\neg\varphi \qquad \varphi \text{ testable} \quad \mathbb{O} \equiv_{R,\varphi} \mathbb{O}' \quad E\ R\ E'}{\mathbb{O}' :_\epsilon E'\mathsf{U}\neg\varphi}\ \text{NegOR}\exists$$

The proof system is sound.

THEOREM 4. *Every sequent $\Delta \Longrightarrow \varphi$ provable in* CIL *is also valid, i.e. $\Delta \models \varphi$.*

We have not investigated completeness and decidability, since their practical importance seems rather limited, and most likely would only hold under overly strong assumptions.

7.2 Derived rules and external premisses

Practical applications of the proof system benefit from using derived rules. For PSS, we rely on the derived rule (UpToBad), whose derivation is given in Figure 2, and which can be viewed as a generalization of the Difference Lemma [35], a.k.a. the Fundamental Lemma [7]:

$$\frac{\mathbb{O}' :_\epsilon E' \quad \mathbb{O}' :_{\epsilon'} \mathsf{F}_{\neg\varphi} \quad \mathbb{O}' \equiv_{R,\varphi} \mathbb{O} \quad E\ R\ E'}{\mathbb{O} :_{\epsilon+\epsilon'} E}\ \text{UpToBad}$$

Likewise, practical applications of the proof system involve establishing external premisses that fall out of the scope of CIL statements. In the PSS example, the premisses are established using standard mathematical reasoning.

More principled and automated methods for establishing external premisses intrinsically depend on the language used to implement oracles. If oracles are given as processes, one would typically rely on process algebraic methods to establish bisimulations; on the contrary, one would use a relational Hoare logic if oracles are given as imperative programs. Likewise, one would use a Hoare logic to establish negligibility statements.

8. PROBABILISTIC SIGNATURE SCHEME

The Probabilistic Signature Scheme [6] is a generic signature scheme that transforms any one-way trapdoor permutation f into a secure signature scheme, and has been adopted as part of the PKCS standard. In this section, we prove that PSS is secure in the random oracle model.

Recall that a signature scheme is composed of a key generation algorithm, a signing algorithm and a signature verification algorithm. Key generation is a probabilistic algorithm that produces a pair (pk, sk) of matching public and private keys—the size of the key is fixed by the security parameter. Signing takes as input the *private* key and a message and returns a valid signature; signing might be deterministic, as in FDH, or probabilistic, as in PSS. The verification algorithm takes as input the *public* key, a message m and a bitstring b and verifies whether b is a valid signature for m; the verification algorithm is deterministic, and yields a boolean value.

A signing scheme is secure against existential forgery under chosen message attack (EF-CMA), if it is not feasible for the adversary to forge a new signature, even when given access to a signing oracle and to the public key. Forgery is modeled by the event ef-cma stating that the adversary has returned a pair $(\mathrm{R}_1, \mathrm{R}_2)$ that is a valid signature, and that has not been produced by the signing oracle:

$$\exists \mathrm{R}_1, \mathrm{R}_2.\ \mathsf{VSig}\ (\mathrm{R}_1, \mathrm{R}_2, m) \wedge\ \mathsf{Fresh}(\mathrm{R}_1, \mathrm{R}_2)$$

where $\mathsf{VSig}\ (\mathrm{R}_1, \mathrm{R}_2)$ and $\mathsf{Fresh}(\mathrm{R}_1, \mathrm{R}_2)$ are the following events:

$$\mathsf{VSig}\ (\mathrm{R}_1, \mathrm{R}_2) = \mathsf{F}_{\lambda((o,q,\_),\_,m).\ o=o_{\mathrm{F}} \wedge q=\mathrm{R}_1 |_{\mathrm{R}_2} \wedge \mathcal{V}(R_1,R_2,m)}$$
$$\mathsf{Fresh}(\mathrm{R}_1, \mathrm{R}_2) = \mathsf{G}_{\lambda((o,q,a),\_,\_).\ \neg(o=\mathsf{sign} \wedge q=\mathrm{R}_1 \wedge a=\mathrm{R}_2)}$$

and $\mathcal{V}$ is the verification algorithm of the scheme—it takes a message, and a forgery candidate, and a memory that contains private data used to produce and check signatures.

The security of any signature scheme S against existential forgery under chosen message attack, under the hypotheses in Δ, can be stated in CIL as $\Delta \Rightarrow \mathsf{S} :_\epsilon$ ef-cma. In the case of PSS the statement is of the form:

$$\mathrm{OW}(f) :_{\epsilon_{\mathbf{OW}}} \mathsf{Invert} \Rightarrow \mathbf{PSS} :_\epsilon \text{ef-cma}$$

where **PSS** is the oracle system that describes PSS and $\mathrm{OW}(f) :_{\epsilon_{\mathbf{OW}}} \mathsf{Invert}$ states the one-wayness of the permutation f, and $\epsilon = \epsilon_{\mathbf{OW}} + \frac{1}{2^{k_2}} + (q_s + q_h)(\frac{q_s}{2^{k_1}} + \frac{q_f+q_g+q_h+q_s}{2^{k_2}})$, and q_s, q_h, q_f, q_g are bounds on the number of sign, $\mathcal{O}_H$, $\mathcal{O}_F$, and $\mathcal{O}_G$ queries performed by the adversary.

The formula $\mathrm{OW}(f) :_{\epsilon_{\mathbf{OW}}} \mathsf{Invert}$ asserts one-wayness of f, using the oracle system $\mathrm{OW}(f)$ and the event Invert defined as follows. The system OW is composed of two oracles: o_{I} and o_{F}. Informally, o_{I} generates the public and inverse keys

$$\dfrac{\dfrac{\dfrac{\mathbb{O}' :_{\epsilon} E'}{\mathbb{O}' :_{\epsilon} E' \wedge \mathsf{G}_{\varphi}}\,\text{UCR} \quad \mathbb{O}' \equiv_{R,\varphi} \mathbb{O} \quad E\ R\ E'}{\mathbb{O} :_{\epsilon} E \wedge \mathsf{G}_{\varphi}}\,\text{NegOR}\forall \qquad \dfrac{\mathbb{O}' :_{\epsilon'} \mathsf{F}_{\neg\varphi} \quad \mathbb{O}' \equiv_{R,\varphi} \mathbb{O}}{\mathbb{O} :_{\epsilon'} \mathsf{F}_{\neg\varphi}}\,\text{NegOR}\diamond}{\mathbb{O} :_{\epsilon'} E \wedge \mathsf{F}_{\neg\varphi}}\,\text{UCR}}{\mathbb{O} :_{\epsilon+\epsilon'} E}\,\text{UCR}$$

Figure 1: Derivation of rule (UpToBad)

for f as well as the challenge y. The oracle o_{F} simply returns its second component. A memory of OW has the form (pk, sk, y, b), where pk, sk are the public and secret keys, y is the challenge and $b \in \{0,1\}$ only serves to make sure that the memory remains unchanged after the first call of o_{I}. As y, pk, sk are generated by o_{I}, the initial memory is irrelevant except that it must ensure $b = 0$. The implementation of o_{I} is as follows:

$$\begin{array}{ll}\lambda x, \_ & \mathsf{if}\ b = 0\ \mathsf{then\ let}\ y \leftarrow \{0,1\}^k\ \mathsf{in}\\ & \quad \mathsf{let}\ (pk, sk) \leftarrow \mathcal{K}\ \mathsf{in}\\ & \quad \mathsf{let}\ b \leftarrow 1\ \mathsf{in}\\ & \quad \mathsf{return}\ (pk, sk, y, b)\\ & \mathsf{else\ return}\ (pk, sk, y, b)\end{array}$$

The event Invert is defined as:

$$\mathsf{F}_{\lambda((o,q,\mathrm{R}),m,\_).\ o=o_{\mathrm{F}} \wedge m=(pk,\_,y,1) \wedge f(pk,q)=y}$$

8.1 Description of PSS

PSS involves hash functions $H : \{0,1\}^* \to \{0,1\}^{k_2}$, and $F : \{0,1\}^{k_2} \to \{0,1\}^{k_0}$, and $G : \{0,1\}^{k_2} \to \{0,1\}^{k_1}$. These functions are modeled as random oracles. In addition, PSS involves a one-way permutation f and its inverse f^{-1} on a group $(\mathcal{G}, \otimes)$; for simplicity, we assume that the elements of $\mathcal{G}$ are represented as elements of $\{0,1\}^k$, where $k = k_0 + k_1 + k_2$. As usual, it is assumed that f is public and f^{-1} is private.

- The probabilistic signature oracle computes the signature of a message m in two steps: first, it samples uniformly a random value r in $\{0,1\}^{k_1}$; then, it computes $w_1 = H(m|r)$, $w_2 = G(w_1) \oplus r$ and $w_3 = F(w_1)$, and returns $f^{-1}(w_1|w_2|w_3)$.

- The signature verification algorithm $\mathcal{V}$ takes as input a bitstring $bs \in \{0,1\}^k$ and a message $m \in \{0,1\}^*$ and checks whether bs is a valid signature for m. It proceeds in two steps: first, it computes $y = f(bs)$ and parses it as $w = w_1|w_2|w_3$ with $w_1 \in \{0,1\}^{k_2}$, $w_2 \in \{0,1\}^{k_1}$ and $w_3 \in \{0,1\}^{k_0}$; then it computes $r = w_2 \oplus G(w_1)$, and checks whether $w_1 = H(m|r)$ and $w_3 = F(w_1)$.

Formally, PSS is modeled by the oracle system $\mathbf{PSS}_0$ s.t.

- a memory m is a tuple of values:

  $$(m.pk, m.sk, m.L_H, m.L_G, m.L_F)$$

  where $m.pk$ and $m.sk$ are the public and private keys, $m.L_H, m.L_G, m.L_F$ are finite functions simulating H, G, F respectively (e.g. $m.L_H \in \{0,1\}^* \rightharpoonup_{\mathsf{fin}} \{0,1\}^{k_2}$). Note that every partial function L has a domain $\mathsf{dom}\ L$ and a range $\mathsf{rng}\ L$, and may be viewed as a set; we often use the notation $L.x$ to denote the union $L \cup \{x\}$, and $[\,]$ to denote the empty partial function;

$$\begin{array}{ll}\mathsf{O}_H(x,m): & \mathsf{if}\ x \in \mathsf{dom}\ L_H\ \mathsf{then\ return}\ (L_H(x), m)\\ & \mathsf{else\ let}\ h \leftarrow \{0,1\}^{k_2}\ \mathsf{in}\\ & \quad \mathsf{return}\ (h, m[L_H := (x,h) \cdot L_H])\\ \mathsf{O}_{\mathsf{sign}}(x,m): & \mathsf{let}\ r \leftarrow \{0,1\}^{k_1}\ \mathsf{in}\\ & \mathsf{let}\ (w_1, m_1) \leftarrow \mathsf{O}_H(x|r, m)\ \mathsf{in}\\ & \mathsf{let}\ (w_2, m_2) \leftarrow \mathsf{O}_G(w_1, m_1)\ \mathsf{in}\\ & \mathsf{let}\ (w_3, m_3) \leftarrow \mathsf{O}_F(w_1, m_2)\ \mathsf{in}\\ & \mathsf{return}\ (f^{-1}(w_1|w_2 \oplus r|w_3), m_3)\end{array}$$

Figure 2: Implementation of oracles in $\mathbf{PSS}_0$. The implementations O_G and O_F are similar to O_H.

- the implementation of the initialization oracle o_{I} is:

  $$\lambda x, m.\ \mathsf{let}\ (pk, sk) \leftarrow \mathcal{K}\ \mathsf{in\ return}\ (pk, sk, [\,], [\,], [\,])$$

- as initial memory one can chose any memory, as the result of o_{I} does not depend on the initial memory;

- the hash oracles $\mathcal{O}_H$, $\mathcal{O}_F$ and $\mathcal{O}_G$ implemented with the functions O_H, O_F and O_G, and the signing oracle sign implemented with $\mathsf{O}_{\mathsf{sign}}$. Their implementations are given in Figure 1, where $m[L_H := L]$ denotes a memory that agrees with m on all components except L_H that gets the value L.

Henceforth, for any bitstring $bs \in \{0,1\}^{k_2+k_1+k_0}$, we denote by $r(bs, m)$ the r-bitstring computed as in the verification oracle in memory m. For $bs \in \{0,1\}^k$ and $\ell \leq \ell' \in [1,k]$, $bs[\ell, \ell']$ denotes the bit-string corresponding to the bits of bs at positions $\ell, \ldots, \ell'$ and $bs[\ell]$ denotes $bs[1, \ell]$, i.e. the prefix of bs of length ℓ.

8.2 Random oracles

The hash functions are modeled as random oracles. More precisely we define an oracle system ROM that simulates the random oracle H, with random answers. The event Guess occurs when the adversary guesses the hash of a value, R_1, i.e. outputs the hash R_2 of R_1 without querying it.

A memory of ROM is a partial mapping L_H. ROM contains an initialization oracle, a finalization oracle and $\mathcal{O}_H$. The latter oracle has the same implementation as in $\mathbf{PSS}_0$ (modulo the type of the memories). The implementations of the initialization and finalization oracles of ROM are as follows:

$$\begin{array}{ll}o_{\mathrm{I}} = \lambda(x,m). & (\mathbf{1}, [\,], [\,], [\,])\\ o_{\mathrm{F}} = \lambda(x,m). & \mathsf{match}\ \mathrm{R}_1 : \{0,1\}^* \mid \mathrm{R}_2 : \{0,1\}^{k_2}\ \mathsf{with}\ x\ \mathsf{in}\\ & \quad \mathsf{return}\ \mathsf{O}_H(\mathrm{R}_2, m)\end{array}$$

The event Guess is $\mathsf{F}_{\mathsf{Guess}_{\mathrm{F}}}$ where $\mathsf{Guess}_{\mathrm{F}}$ is:

$$\begin{array}{l}\lambda((o,q,\_), m, m').\exists(\mathrm{R}_1, \mathrm{R}_2).\\ o = o_{\mathrm{F}} \wedge q = \mathrm{R}_1 \mid \mathrm{R}_2 \wedge m'.L_H(\mathrm{R}_1) = \mathrm{R}_2 \wedge \mathrm{R}_1 \notin \mathsf{dom}\ m.L_H\end{array}$$

For every oracle o, let ϵ_o denote:

$$\Pr[\mathsf{O}_o(q,m) = (a,m') \wedge \mathsf{Guess}_{\mathrm{F}}((o,q,a), m, m')]$$

$\mathsf{C}^{\rightarrow}_{c_{\mathrm{I}}}(x, m_c)$: return $((o_{\mathrm{I}}, \mathbf{1}), m_c)$

$\mathsf{C}^{\leftarrow}_{c_{\mathrm{I}}}(x, (o, q, a), m_c)$: let $(pk, sk) \leftarrow \mathcal{K}$ in return $(pk, sk, [\,], [\,])$

$\mathsf{C}^{\rightarrow}_{c_{sign}}(x, m_c)$: let $r \leftarrow \{0,1\}^{k_1}$ in return $((\mathcal{O}_H, x|r), m_c)$

$\mathsf{C}^{\leftarrow}_{c_{sign}}(x, (o, q, a), m_c)$: match $x|w_1$ with q in let $(w_2, m'_c) \leftarrow \mathsf{O}_G(w_1, m_c)$ in let $(w_3, m''_c) \leftarrow \mathsf{O}_F(w_1, m'_c)$ in return $(f^{-1}(w_1|w_2 \oplus r|w_3), m''_c)$

$\mathsf{C}^{\leftarrow}_{o_{\mathrm{F}}}(x, m_c)$: match $\mathrm{R}_1|x' : \{0,1\}^k$ with x in match $w_1|w_2|w_3$ with $f(pk, x')$ in let $(g, m'_c) \leftarrow \mathsf{O}_G(w_1, m_c)$ in let $r \leftarrow w_2 \oplus g$ in return $(\mathrm{R}_1|r, m'_c)$

Figure 4: Implementation of the forward and backward implementations in $\mathbb{C}_{\mathrm{ROM}}$

Note that $\epsilon_o = 0$ if $o \neq o_{\mathrm{F}}$ and $\epsilon_{o_{\mathrm{F}}} = 2^{-k_2}$. Therefore, using rule (FAIL), we have ROM $:_{2^{-k_2}}$ Guess.

We conclude this section with a mild subtlety. In order to apply to ROM contexts who may contain procedures that do not call $\mathcal{O}_H$, we add a "dummy" oracle $o_d = \lambda(x, m).\,(\mathbf{1}, m)$. Of course the "dummy" oracle does not invalidate the validity of the derivation above.

8.3 Formal proof

The proof tree of PSS is given in Figure 3. We briefly explain the proof tree below, in a bottom-up approach.

We can use the rule (UR) to perform a case analysis on $\mathrm{R}_1|r(\mathrm{R}_2, m) \in m.L_H$. It yields two new events $\mathsf{ef\text{-}cma}_1$ and $\mathsf{ef\text{-}cma}_2$, by respectively adding to the F-formula the conjuncts $\mathrm{R}_1|r(\mathrm{R}_2, m) \notin m.L_H$ and $\mathrm{R}_1|r(\mathrm{R}_2, m) \notin m.L_H$. Since for every trace τ, if $\mathsf{ef\text{-}cma}(\tau)$ then $\mathsf{ef\text{-}cma}_1(\tau) \vee \mathsf{ef\text{-}cma}_2(\tau)$, (UR) yields:

$$\left.\begin{array}{lll} \mathbf{PSS}_0 & :_{2^{-k_2}} & \mathsf{ef\text{-}cma}_1 \\ \mathbf{PSS}_0 & :_{\epsilon - 2^{-k_2}} & \mathsf{ef\text{-}cma}_2 \end{array}\right\} \Longrightarrow \mathbf{PSS}_0 :_{\epsilon} \mathsf{ef\text{-}cma} \quad (1)$$

The second branch $\mathbf{PSS}_0 :_{2^{-k_2}} \mathbf{ef\text{-}cma}_1$ is derived from the properties of random oracles. The proof tree is:

$$\dfrac{\dfrac{}{\mathrm{ROM} : \mathsf{Guess}}\ (\mathrm{FAIL})}{\mathbf{PSS}_0 :_{2^{-k_2}} \mathsf{ef\text{-}cma}_1}\ (\mathrm{NegSUB})$$

To apply (negSUB) we define a ROM-context $\mathbb{C}_{\mathrm{ROM}}$ such that $\mathbf{PSS}_0 = \mathbb{C}_{\mathrm{ROM}}[\mathrm{ROM}]$[1]. A memory of $\mathbb{C}_{\mathrm{ROM}}$ has the form (pk, sk, L_G, L_F). Its initial memory is the same as the initial memory of $\mathbf{PSS}_0$ after projecting out L_H. The procedures of $\mathbb{C}_{\mathrm{ROM}}$ are named c_{I}, c_{F}, c_f, c_g and c_{sign}. The forward and backward implementations of c_{I}, c_{F} and c_{sign} are given in Figure 4. The forward implementations of c_f and c_g simply call the "dummy" oracle o_d of ROM and do not modify the context memory. Their backward implementations call the implementations O_G and O_F to compute the requested hashes.

Next, we define the oracle system $\mathbf{PSS}_1$ (see Figure 5). In this oracle system, we compute $F(h)$ and $G(h)$ each time h is produced by H. To be consistent with $\mathbf{PSS}_0$, we introduce

[1]More precisely, this equality holds modulo tuple associativity which can be captured using a bisimulation.

$\mathsf{O}_H(x, m)$:
if $x \in \mathsf{dom}\, L_H$ then return $(L_H(x), m)$
else let $w_1|w_2|w_3 \leftarrow \{0,1\}^{k_2} \times \{0,1\}^{k_1} \times \{0,1\}^{k_0}$ in
let $m_1 \leftarrow m[L_H := (x, w_1) \cdot L_H]$ in
let $m_2 \leftarrow \mathsf{Upd}(w_1, w_2 \oplus r, G, m_1)$ in
let $m_3 \leftarrow \mathsf{Upd}(w_1, w_3, F, m_2)$ in
return (w_1, m_3)

where $\mathsf{Upd} = \lambda(x, w, X, m).$ if $x \in L_X, L'_X$ then (w, m) else $(w, m[L'_X := (x, w) \cdot L'_X])$

$\mathsf{O}_G(x, m)$:
if $x \in \mathsf{dom}\, L_G$ then return $(L_G(x), m)$
else if $x \in \mathsf{dom}\, L'_G$ then
return $(L'_G(x), m[L'_G \xrightarrow{x} L_G])$
else let $g \leftarrow \{0,1\}^{k_1}$ in
return $(g, m[L_G := (x, g) \cdot L_G])$

where $m[L'_G \xrightarrow{x} L_G] =$
$m[L_G := (x, L'_G(x)) \cdot L_G, L'_G := L'_G \setminus (x, L'_G(x))]$

$\mathsf{O}_{\mathsf{sign}}(x, m)$: let $r \leftarrow \{0,1\}^{k_1}$ in let $(w_1, m_1) \leftarrow \mathsf{O}_H(x|r, m)$ in let $(w_2, m_2) \leftarrow \mathsf{O}_G(w_1, m_1)$ in let $(w_3, m_3) \leftarrow \mathsf{O}_F(w_1, m_2)$ in return $(f^{-1}(w_1|w_2 \oplus r|w_3), m_3)$

Figure 5: Implementation of oracles in $\mathbf{PSS}_1$

two new variables L'_F and L'_G that have the same type as L_F and L_G. The idea is to store the pre-computed hash values in L'_F and L'_G, and to transfer them from L'_F (resp. L'_G) to L_F (resp. L_G) once the values are requested to $\mathcal{O}_G$ and $\mathcal{O}_G$ respectively. This is a case of *eager sampling* that we handle with our determinization techniques. Indeed, we can show that we have $\mathbf{PSS}_1 \leq_{\mathrm{det},\gamma} \mathbf{PSS}_2$, where γ is as follows. Consider a memory m of $\mathbf{PSS}_1$. Let $X = (\mathsf{rng}\ L_H) \setminus (\mathsf{dom}\ L'_G)$ (resp. $Y = (\mathsf{rng}\ L_H) \setminus (\mathsf{dom}\ L'_F)$). Then, $\gamma(m)$ is the uniform distribution over all pairs (L'_G, L'_F) with L'_G (resp. L'_F) a mapping (viewed as a set) in $X \to \{0,1\}^{k_1}$ (resp. $Y \to \{0,1\}^{k_0}$). Using rule (NegDET), we have:

$$\mathbf{PSS}_1 :_{\epsilon - 2^{-k_2}} \mathsf{ef\text{-}cma}_2 \Longrightarrow \mathbf{PSS}_0 :_{\epsilon - 2^{-k_2}} \mathsf{ef\text{-}cma}_2 \quad (2)$$

Next, we define the oracle system $\mathbf{PSS}_2$. We do a number of changes w.r.t. $\mathbf{PSS}_1$:

1. We anticipate the computation of $F(h)$ and $G(h)$ regardless of whether they have been previously computed or not. This makes the new system differ from the previous $\mathbf{PSS}_1$ in case H produces a hash value that has been either produced before for a different input or directly queried by the adversary or by the signing oracle.

2. We introduce a new variable y whose value is uniformly sampled in $\{0,1\}^k$. This prepares for the one-way challenge.

3. In the implementation of O_H, we modify how $w_1|w_2|w_3$ is determined by sampling a value u in $\{0,1\}^k$ and computing $w_1|w_2|w_3$ as $f(u) \otimes y$, where $\otimes$ is the inner law of group $\mathcal{G}$. Since f is a permutation, both ways

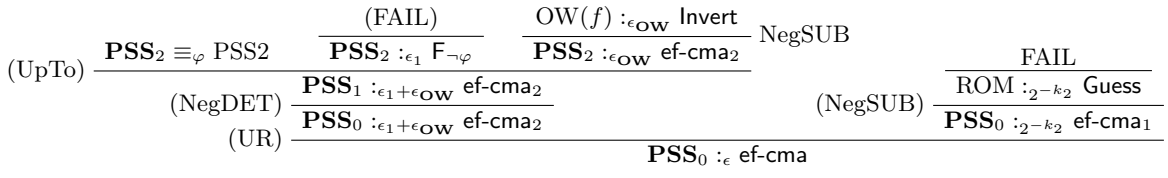

Figure 3: Proof tree for PSS

of computing $w_1|w_2|w_3$ are equivalent. In the signing oracle, we do not perform the group operation and compute $w_1|w_2|w_3$ as $f(u)$.

4. We introduce a list L_u that allows us find the value u from which originates a H-hash computed as the k_2 prefix of $f(u) \otimes y$.

The implementations of the oracles of $\mathbf{PSS}_2$ are given in Figure 6. Now, let the predicate φ be defined on triples $((o, q, a), m, m')$ as the conjunction of the clauses:

- if $o = \mathcal{O}_H \wedge q \notin m.L_H$ then $a \notin \mathsf{dom}\ (m.L_F \cup m.L'_F \cup m.L_G \cup m.L'_G)$
- if $o = \mathcal{O}_{\mathsf{sign}}$ then $w_1 \notin \mathsf{dom}\ (m.L_F \cup m.L'_F \cup m.L_G \cup m.L'_G)$
- if $o = \mathcal{O}_{\mathsf{sign}}$ then $\forall (w_1, g) \in m'.L_G.\ q \mid (w_2 \oplus g) \notin \mathsf{dom}\ m.L_H$

where $w_1 = f(pk, a)[1, k_2]$ and $w_2 = f(pk, a)[k_2, k_1 + 1]$.

Using (FAIL), we can establish $\mathbf{PSS}_2 :_{\epsilon_1} \mathsf{F}_{\neg\varphi}$, with

$$\epsilon_1 = (q_s + q_h)(\frac{q_s}{2^{k_1}} + \frac{q_f + q_g + q_h + q_s}{2^{k_2}}).$$

Indeed, a, respectively w_1 and w_2, is a freshly uniformly sampled value in $\{0,1\}^{k_2}$, resp. $\{0,1\}^{k_2}$ and $\{0,1\}^{k_1}$, hence, the probability of forcing $\neg\varphi$ in a single call to an oracle is $(q_f + q_g + q_h + q_s)2^{-k_2}$, resp. $(q_h + q_s)2^{-k_1}$.

Moreover, $\mathbf{PSS}_1$ and $\mathbf{PSS}_2$ are R-bisimilar until φ, where R is such that $m\, R\, m'$ iff m and m' coincide on their common components, namely $L_H, L_G, L'_G, L_F, L'_F, pk, sk$. Using the rule (UpToBad), we obtain:

$$\left.\begin{array}{lll} \mathbf{PSS}_2 & :_{\epsilon_1} & \mathsf{F}_{\neg\varphi} \\ \mathbf{PSS}_2 & \equiv_{R,\varphi} & \mathbf{PSS}_1 \\ \mathbf{PSS}_2 & :_{\epsilon_{\mathbf{OW}}} & \mathsf{ef\text{-}cma}_2 \end{array}\right\} \Rightarrow \mathbf{PSS}_1 :_{\epsilon - 2^{-k_2}} \mathsf{ef\text{-}cma}_2 \quad (3)$$

The oracle implementations of $\mathbf{PSS}_2$ do not use the trapdoor key sk. Therefore, it is easy to write $\mathbf{PSS}_2$ as a context $\mathbb{C}$ applied to $\mathrm{OW}(f)$, i.e. $\mathbf{PSS}_2 = \mathbb{C}[\mathrm{OW}(f)]$. However, we want to choose the context $\mathbb{C}$ such that:

$$\mathsf{ef\text{-}cma}_2 \circ \mathbb{C} \Rightarrow \mathsf{Invert}$$

To this end, we define the forward implementation of the finalization procedure of $\mathbb{C}$ as follows:

$$\begin{array}{ll} \mathsf{C}^{\rightarrow}_{c_{\mathrm{F}}}(x, m_c): & \mathsf{match}\ \mathrm{R}_1|\mathrm{R}_2 : \{0,1\}^k\ \mathsf{with}\ x\ \mathsf{in} \\ & \mathsf{match}\ w_1|w_2|w_3\ \mathsf{with}\ f(pk, \mathrm{R}_2)\ \mathsf{in} \\ & \mathsf{let}\ (g, m'_c) \leftarrow L_G(w_1)\ \mathsf{in} \\ & \mathsf{let}\ r \leftarrow w_2 \oplus g\ \mathsf{in} \\ & \mathsf{let}\ (u, w'_1) \leftarrow L_u(\mathrm{R}_1, r)\ \mathsf{in} \\ & \mathsf{return}\ \mathrm{R}_2 \oslash u \end{array}$$

When $\mathsf{ef\text{-}cma}_2$ is satisfied, we have $w'_1 = L_H(w_2) = w_1$, and $w_3 = L_F(w_1)$ and $w_1|w_2|w_3 = f(u) \otimes y$. Since f is homomorphic, it entails that $u \oslash \mathrm{R}_2 = f(sk, y)$, where $\oslash$ is the inverse operation of $\otimes$ in group $\mathcal{G}$. Hence, $f(pk, \mathrm{R}_2 \oslash u) = y$ and Invert is satisfied. Therefore, we have:

$$\mathrm{OW}(f) :_{\epsilon_{\mathbf{OW}}} \mathsf{Invert} \Longrightarrow \mathbf{PSS}_2 :_{\epsilon_{\mathbf{OW}}} \mathsf{ef\text{-}cma}_2 \quad (4)$$

which concludes the proof.

9. RELATED WORK

Impagliazzo and Kapron [26] were the first to develop a logic to reason about indistinguishability. Their logic is built upon a more general logic whose soundness relies on non-standard arithmetic; they show the correctness of a pseudo-random generator, and that next-bit unpredictability implies pseudo-randomness. Recently, Zhang [37] developed a similar logic on top of Hofmann's SLR system [25], and reconstructs the examples of [26]. These logics have a limited applicability because of their lack of support for oracles or adaptive adversaries. capture the standard reasoning patterns for reasoning about cryptographic schemes.

Independently, Corin and den Hartog [15] prove semantic security of ElGamal using a variant of a general purpose probabilistic Hoare logic; again, their logic does not consider oracles. Our logic CIL generalizes these frameworks by accounting for oracles and adaptive adversaries. CIL also generalizes the specialized Hoare-like logic of Courant *et al* [17]: this logic supports automated proofs of IND-CPA for encryption schemes in the random oracle model. However, it is not clear how to use it to show security of OAEP.

There have been similar efforts to develop computational logics for protocols. One prominent example of such a logic is Protocol Composition Logic (PCL) of Datta et al. [20], which has been applied successfully to the IEEE 802.11i wireless security standard and the IETF GDOI standard. PCL is a computationally sound Hoare-like logic for indistinguishability; one important difference is that, being focused on protocols, PCL does not provide support for standard cryptographic constructions such as one-way functions. Motivated by work on PCL, Halpern [24] considers a first-order logic, where $\varphi \rightarrow \psi$ means that $\Pr[\psi|\varphi]$ is overwhelming. He provides a complete axiomatization and shows how a qualitative proof of asymptotic security can be converted to a qualitative proof of concrete security. It does not seem straightforward to use Halpern's logic for adaptative security definitions and proofs.

Observational equivalence is a standard tool from programming language theory and it can been used effectively to express or approximate security properties such as secrecy, authenticity, or even anonymity. Abadi and Gordon [1] use observational equivalence to formulate secrecy in the spi-calculus, and define a sound equational theory for proving observational equivalence between two processes;

$\mathcal{O}_H(x,m)$:
if $x \in \mathsf{dom}\ L_H$ then return $(L_H(x), m)$
else
if $|x| < k_1 + 1$ then let $w_1 \leftarrow \{0,1\}^{k_2}$ in
return $(w_1, m[L_H := (x, w_1) \cdot L_H])$
else
let $u \leftarrow \{0,1\}^k$ in
let $w \leftarrow f(pk, u) \otimes y$ in
match $w_1|w_2|w_3$ with w in
match $M|r : \{0,1\}^{k_1}$ with x in
let $m' \leftarrow m[L_H := L_H \cdot (x, w_1),$
$L'_F := L'_F \cdot (w_1, w_3),$
$L'_G := L'_G \cdot (w_1, w_2 \oplus r)$
$L_u := L_u \cdot (M, r, u, w_1)]$ in
return (w_1, m')

$\mathcal{O}_{\mathsf{sign}}(x,m)$:
let $r \leftarrow \{0,1\}^{k_1}$ in
let $u \leftarrow \{0,1\}^k$ in
let $w \leftarrow f(pk, u)$ in
let $w_1|w_2|w_3 \leftarrow w$ in
let $w'_2 \leftarrow w_2 \oplus r$ in
let $m' \leftarrow m[L_H := (x|r, w_1), L'_F := L'_F \cdot (w_1, w_3),$
$L'_G := L'_G \cdot (w_1, w'_2)]$ in
return (u, m')

Figure 6: Implementation of signing oracle and $\mathcal{O}_H$ oracle in $\mathbf{PSS}_2$

using the equational theory, they establish the security of common protocols. While the spi-calculus relies on a symbolic model of cryptography, Mitchell *et al* [32] develop a theory of observational equivalence in a process algebra for probabilistic polynomial time computation, and show how to use the equivalence to reason about cryptographic primitives. In particular, they develop a proof system for bisimulation for their process algebra. Segala et al. [13, 34] develop a model of probabilistic I/O-automata adapted to security proofs and a corresponding approximated bounded probabilistic simulation relations.

Observational equivalence in the symbolic and computational models can be related formally via computational soundness results, which show that the security guarantees that can be derived from reasoning in symbolic models remain meaningful in the computational model. In their seminal work, Abadi and Rogaway [2] prove that symbolic observational equivalence is a sound abstraction of computational observational equivalence. This line of work has been extended in many other works, e.g. [31, 30, 14]. However, there are known limitations to computational soundness, and several authors have attempted to circumvent them by developing proof systems or type systems that directly enforce computational non-interference, see e.g. [18, 27], which is closely related to computational observational equivalence.

A more direct approach is to develop a proof system for reasoning about observational equivalence directly in the computational model. One prominent example of this approach is Blanchet's CryptoVerif [8], a semi-automatic tool that allows carrying exact security proofs following the game-based technique. CryptoVerif allows reasoning both about primitives and protocols, and has been used successfully to prove the security of many protocols, including Kerberos [9] and of Full Domain Hash [10]—for the non-optimal bound. It is difficult to assess CryptoVerif ability to handle automatically more complex cryptographic proofs, e.g. for schemes such as OAEP and PSS, or even for Coron's improved bound for FDH [16]. In addition, it remains challenging to generate independently verifiable proofs from successful runs of CryptoVerif, see however [22].

Universal composability [11] is a paradigm for the design of protocols using a very general simulation-based definition. As discussed by Canetti [12], the UC framework can serve as a basis for composable formal systems for security analysis, and notes that such an approach is partly realized in the work of Backes *et al* [4]. One drawback of tying a formal approach to the UC framework are the inherent limitations imposed by the strength of UC security definitions. This is discussed in detail by Datta *et al* [19].

One central motivation for developing rigorous logics for cryptography is that they can be formalized in a proof assistant and then used to mechanically check the correctness of cryptographic schemes. Machine checked proofs have been suggested as a means to improve confidence in cryptographic proofs, notably by Bellare and Rogaway [7] and with more emphasis by Halevi [23]. Yet only a few cryptographic primitives have been machine-checked. Backes and co-workers [3] are developing a comprehensive framework for machine-checking cryptographic proofs using the Isabelle proof assistant. In a similar spirit, Barthe *et al* [5] are developing CertiCrypt, which provides support for formalizing game-based proofs in the Coq proof assistant. CIL and CertiCrypt are complementary and form an excellent match—we have carried simultaneously in both systems the proof of IND-CCA-SE of OAEP reported in [**?**].

10. CONCLUSION

Computational Indistinguishability Logic (CIL) is a general logic that captures in a small set of rules many common reasoning patterns in cryptographic proofs. We have focused on a core logic and shown how many cryptographic techniques (proofs by reduction, simulations, lazy/eager sampling etc) are closely related to foundational notions in programming languages semantics and process algebra (contexts, observational equivalence, determinization). There are many relevant extensions to the core logic, e.g. conditional reasoning and iterated oracle systems that are useful for dealing with protocols.

One priority for future work is to integrate CIL with a language-specific framework for cryptographic proofs. We have formalized most of CIL in Coq, and plan to integrate our formalization to CIL. We expect that CIL will significantly improve proof automation and proof reuse in CertiCrypt. One relevant issue w.r.t. proof automation is to develop heuristics for deciding whether a given relation is a bisimulation up to. The Relational Hoare Logic of CertiCrypt conveniently supports reasoning about bisimulation up to, and we have obtained promising results in the design of automated methods to establish the validity of relational Hoare statements.

Our long term objective is to develop sound and practical verification tools for cryptography. We believe that programming languages theory and process algebra will not only help establishing sound foundations, but also practical techniques for these tools, and will eventually have a lasting impact on provable security.

11. REFERENCES

[1] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999.

[2] Martín Abadi and Philipp Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[3] Michael Backes, Mathias Berg, and Dominique Unruh. A formal language for cryptographic pseudocode. In *Proceedings of LPAR'08*, pages 353–376. Springer-Verlag, 2008.

[4] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM Conference on Computer and Communications Security*, pages 220–230. ACM, 2003.

[5] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *Proceedings of POPL'09*, pages 90–101. ACM Press, 2009.

[6] Mihir Bellare and Philipp Rogaway. The exact security of digital signatures – How to sign with RSA and Rabin. In *Proceedings of EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer-Verlag, 1996.

[7] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Proceedings of EUROCRYPT'06*, pages 409–426, 2006.

[8] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *IEEE Symposium on Security and Privacy*, pages 140–154. IEEE Computer Society, 2006.

[9] Bruno Blanchet, Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay. Computationally sound mechanized proofs for basic and public-key Kerberos. In *Proceedings of ASIACCS'08*, pages 87–99. ACM, 2008.

[10] Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. In *Advances in Cryptology – CRYPTO'06*, volume 4117 of *Lecture Notes in Computer Science*, pages 537–554. Springer-Verlag, 2006.

[11] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[12] Ran Canetti. Composable formal security analysis: Juggling soundness, simplicity and efficiency. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2008.

[13] Ran Canetti, Ling Cheung, Dilsun Kirli Kaynar, Moses Liskov, Nancy A. Lynch, Olivier Pereira, and Roberto Segala. Analyzing security protocols using time-bounded task-pioas. *Discrete Event Dynamic Systems*, 18(1):111–159, 2008.

[14] Hubert Comon-Lundh and Véronique Cortier. Computational soundness of observational equivalence. In *Proceedings of CCS'08*, pages 109–118. ACM Press, October 2008.

[15] Ricardo Corin and Jerry den Hartog. A probabilistic Hoare-style logic for game-based cryptographic proofs. In *Proceedings of ICALP'06*, volume 4052 of *LNCS*, pages 252–263, 2006.

[16] Jean Sébastien Coron. On the exact security of Full Domain Hash. In *Proceedings of CRYPTO'00*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer-Verlag, 2000.

[17] Judicaël Courant, Marion Daubignard, Cristian Ene, Pascal Lafourcade, and Yassine Lakhnech. Towards automated proofs for asymmetric encryption schemes in the random oracle model. In *Proceedings of CCS'08*, pages 371–380. ACM Press, 2008.

[18] Judicaël Courant, Cristian Ene, and Yassine Lakhnech. Computationally sound typing for non-interference: The case of deterministic encryption. In *Proceedings of FSTTCS'07*, volume 4855 of *Lecture Notes in Computer Science*, pages 364–375. Springer, 2007.

[19] Anupam Datta, Ante Derek, John C. Mitchell, Ajith Ramanathan, and Andre Scedrov. Games and the impossibility of realizable ideal functionality. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 360–379. Springer, 2006.

[20] Anupam Datta, Ante Derek, John C. Mitchell, and Bogdan Warinschi. Computationally sound compositional logic for key exchange protocols. In *Proceedings of CSFW'06*, pages 321–334. IEEE Computer Society, 2006.

[21] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[22] Jean Goubault-Larrecq. Towards producing formally checkable security proofs, automatically. In *Proceedings of CSF'08*, pages 224–238. IEEE Computer Society, 2008.

[23] Shai Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181, 2005.

[24] Joseph Y. Halpern. From qualitative to quantitative proofs of security properties using first-order conditional logic. In *Proceedings of AAAI'08*, pages 454–459, 2008.

[25] Martin Hofmann. A Mixed Modal/Linear Lambda Calculus with Applications to Bellantoni-Cook Safe Recursion. In *Proceedings of CSL'97*, pages 275–294, 1997.

[26] Russell Impagliazzo and Bruce M. Kapron. Logics for reasoning about cryptographic constructions. *Journal of Computer and Systems Sciences*, 72(2):286–320, 2006.

[27] Peeter Laud. On the computational soundness of

cryptographically masked flows. In *Proceedings of POPL 2008*, pages 337–348. ACM, 2008.

[28] Ueli Maurer. Random systems: Theory and applications. In Yvo Desmedt, editor, *ICITS 2007*, volume 4883 of *Lecture Notes in Computer Science*, pages 44–45. Springer-Verlag, 2009.

[29] Ueli Maurer, Krzysztof Pietrzak, and Renato Renner. Indistinguishability amplification. In Alfred Menezes, editor, *Advances in Cryptology — CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 130–149. Springer-Verlag, August 2007.

[30] Daniele Micciancio and Saurabh Panjwani. Adaptive security of symbolic encryption. In Joe Kilian, editor, *Proceedings of TCC'05*, volume 3378 of *Lecture Notes in Computer Science*, pages 169–187. Springer-Verlag, 2005.

[31] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proceedings of TCC'04*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.

[32] John C. Mitchell, A. Ramanathan, Andre Scedrov, and Vanessa Teague. Probabilistic Polynominal-Time process calculus and security protocol analysis. In *Proceedings of LICS'01*, pages 3–8. IEEE Computer Society, 2001.

[33] David Nowak. A framework for game-based security proofs. In *Proceedings of ICS'07*, volume 4861, pages 319–333. Springer-Verlag, 2007.

[34] Roberto Segala and Andrea Turrini. Approximated computationally bounded simulation relations for probabilistic automata. In *CSF*, pages 140–156. IEEE Computer Society, 2007.

[35] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint 2004/332, 2004.

[36] Jacques Stern. Why provable security matters? In *Advances in Cryptology – EUROCRYPT'03*, volume 2656 of *Lecture Notes in Computer Science*, pages 449–461. Springer-Verlag, 2003.

[37] Yu Zhang. The computational SLR: a logic for reasoning about computational indistinguishability. IACR ePrint Archive 2008/434, 2008. Also in Proc. of Typed Lambda Calculi and Applications 2009.