Foundations for Cryptographic Reductions in CCSA Logics

David Baelde, Adrien Koutsos, Justine Sauvage

Foundations for Cryptographic Reductions in CCSA Logics
(long version)

David Baelde
Univ Rennes, CNRS, IRISA
Rennes, France

Adrien Koutsos
Inria
Paris, France

Justine Sauvage
Inria
Paris, France

ABSTRACT

The Computationally Complete Symbolic Attacker (CCSA) approach to security protocol verification relies on probabilistic logics to reason about the interaction traces between a protocol and an arbitrary adversary. The proof assistant Squirrel implements one such logic. CCSA logics come with cryptographic axioms whose soundness derives from the security of standard cryptographic games, e.g. PRF, EUF, IND-CCA. Unfortunately, these axioms are complex to design and implement; so far, these tasks are manual, ad hoc and error-prone. We solve these issues by providing a formal and systematic method for deriving axioms from cryptographic games. Our method relies on synthesizing an adversary against some cryptographic game, through the notion of bi-deduction. Concretely, we define a rich notion of bi-deduction, justify how to use it to derive cryptographic axioms, provide a proof system for bi-deduction, and an automatic proof-search method which we implemented in Squirrel.

KEYWORDS

Logic, Security Protocols, Cryptographic Games, Provable Security

1 INTRODUCTION

Computer systems are being used for increasingly many applications in our digitalized societies: messaging, payments, voting, etc. All these applications involve communication protocols which use cryptographic primitives to ensure various security properties. Proving the security of protocols is both notoriously difficult and very important, given their wide deployment and their use in critical applications. It has been the topic of intense research over the past decades. In this paper, we are concerned with obtaining formal proofs of protocol designs. We thus consider high-level models, leaving aside the implementation details even though they can be the source of many exploits at the software and hardware levels (e.g. [1, 35]). We also ignore the inner workings of cryptographic primitives, treating them as black boxes ensuring some functionality while preventing unwanted behaviors, which can of course hide other vulnerabilities (e.g. [3]). Importantly, these abstractions do not trivialize the analysis of protocols (e.g. see [4, 17] for striking attacks found at this level of abstraction, and [11] for a state of the art).

We seek to obtain security guarantees in the cryptographers' standard model for *provable security*, also known as the *computational model*. In that setting, protocols and adversaries are modeled as Polynomial-time Probabilistic Turing Machines (PPTMs). We consider a general class of security properties, expressed as indistinguishability between two protocols — typically, a real protocol and an ideal one corresponding to an obviously correct but non-practical implementation of the target application. To prove such protocol indistinguishabilities, we rely on *cryptographic assumptions*, which are properties of cryptographic primitives, also expressed as indistinguishabilities. The notion of *cryptographic game*, articulating the interactions of some unknown adversary with either a full protocol or an isolated primitive, thus plays a central role.

Proofs in the computational model typically proceed by "game hopping" [44]: one repeatedly reduces the indistinguishability of some game to that of another one (up to a negligible probability). For instance, protocol indistinguishability may be reduced in this way to the lower-level indistinguishabilities on primitives. This is generally too complex to be fully formalized, and the usual pen-and-paper proofs leave gaps that often hide errors. To overcome this difficulty, mechanized verification tools have been developed. For instance, CryptoVerif [18] automates common game-hopping transformations, Owl [29] allows to prove integrity and confidentiality properties by typing, while other tools (Easycrypt [15], CryptHOL [16], SSProve [2]) allow reasoning through (probabilistic and relational) program logics.

More recently, Bana and Comon [10] have proposed an alternative approach to prove the computational security of a protocol, which abstracts away the quantitative details about probabilities and security parameters, allowing to reason purely symbolically on the structure of messages. They design a first-order logic — the CCSA logic — where terms are interpreted as PPTMs computing messages, and a predicate $\vec{u} \sim \vec{v}$ corresponds to computational indistinguishability. The security properties of cryptographic primitives are reformulated as formulas in that logic — called *cryptographic axioms* — whose soundness directly derives from the indistinguishability of the associated cryptographic games: in other words, cryptographic assumptions become logical axioms. The security of a protocol is then expressed as a formula, which must be shown to be a logical consequence of the relevant cryptographic axioms. Several proofs have been carried out using this approach, first by hand [9, 22, 36, 43] and later using the proof assistant Squirrel [5, 6, 8, 26], which relies on enriched CCSA logics that notably internalize the notions of protocol and execution traces. Initially proposed as a CCSA *meta-logic* [5], the logic behind Squirrel is now a more general, *higher-order* CCSA logic [8].

Example 1. *The PRF cryptographic assumption on a keyed hash function* h *roughly states that hashes* $\mathsf{h}(\_, \mathsf{k})$ *using a secret key* k *are indistinguishable from random values. More precisely, it can be expressed as the indistinguishability of two games* G_0 *and* G_1*, where the key* k *is initially sampled, and the adversary is provided with two oracles: the* hashing *oracle allows to compute hashes of chosen messages in both* G_0 *and* G_1*; the* challenge *oracle returns the hash of its input in* G_0*, but a fresh sampling in* G_1*. To avoid irrelevant distinguishing attacks, both oracles reject inputs that have already been used. This game is an equivalent variant of the standard one for PRF, which will facilitate its translation into a CCSA axiom.*

In the original CCSA logic, random samplings of the protocol are represented by special constants called names*, and other function symbols represent PPTMs that cannot access these samplings. The PRF assumption is expressed through the following axiom scheme [22]:*

$$\vec{u}, \mathsf{h}(v, \mathsf{k}) \quad \sim \quad \vec{u}, \mathsf{if}\ (\bigwedge_{m \in H} v \neq m)\ \mathsf{then}\ \mathsf{n}_{fresh}\ \mathsf{else}\ \mathsf{h}(v, \mathsf{k})$$

Here, $\vec{u}$ and v must be closed terms (i.e. terms without free variables); H is the set of all terms m such that $\mathsf{h}(m, \mathsf{k})$ is a subterm of $\vec{u}, v$; the name k must only occur as a hashing key; and the name n_{fresh} must have a single occurrence. All these conditions guarantee that there exists a PPTM acting as an adversary in the PRF game which computes $\vec{u}$ and v, using the game's hashing oracle to compute $\mathsf{h}(m, \mathsf{k})$ for each $m \in H$. From this we can build another adversary that tests whether $v \neq m$ for all $m \in H$, calls the challenge oracle on v if this is the case, and otherwise calls the hashing oracle on v. It will thus obtain $\mathsf{h}(v, \mathsf{k})$ in G_0 and the above if-then-else term in G_1. If a PPTM could distinguish the two sides of the PRF axiom with non-negligible probability, we would thus have an adversary with non-negligible advantage in the PRF game. That is, the cryptographic assumption implies the soundness of our axiom scheme.

In the enriched CCSA logics behind Squirrel, formulating axiom schemes becomes much more complex because some formulas and terms are recursively defined over the execution trace, which makes it impossible to determine statically e.g. the subterms of the form $\mathsf{h}(m, \mathsf{k})$. Much of the complexity of designing these logics goes into determining precise-enough over-approximations of sets of occurrences, and incorporating these approximations into soundness arguments. These issues are exacerbated with more complex cryptographic assumptions, e.g. indistinguishability under chosen-ciphertext (IND-CCA$_1$) [8, App. D]. This source of complexity has caused errors both in the theory and implementation of Squirrel, which currently relies on tedious verifications of the various syntactic side conditions, and does not allow the user to control approximations. Moreover, adding new cryptographic axioms to Squirrel currently requires writing OCaml code (the language Squirrel is implemented in), a time-consuming task that requires an in-depth understanding of both the theoretical framework and its implementation, putting it out-of-reach of all but the most expert users. This is the problem we aim to address in this work.

Soundness of Cryptographic Axioms. There is a fundamental connection between cryptographic games and the corresponding CCSA axioms: a CCSA axiom $\vec{u}_0 \sim \vec{u}_1$ is valid under some cryptographic assumption represented by a game $\mathsf{G} = (\mathsf{G}_0, \mathsf{G}_1)$ whenever the terms $\vec{u}_0$ and $\vec{u}_1$ can be computed by an adversary interacting with G. More precisely, there must exist a *single* PTIME program $\mathcal{S}$ (the *simulator*) such that $\mathcal{S}$ produces $\vec{u}_i$ when interacting with G_i for both $i \in \{0, 1\}$. Until now, all soundness proofs of CCSA axioms have been proved manually, by exhibiting (on paper) such a simulator $\mathcal{S}$. This has been done in an ad hoc fashion, for a few cryptographic axioms and games (IND-CCA$_1$, EUF-CMA…).

Interestingly, some work has already been initiated to mechanize the proofs of existence of simulators, albeit with a different aim in mind. In [6], the notion of bi-deduction is introduced: a judgment $(\vec{v}_0, \vec{v}_1) \triangleright (\vec{u}_0, \vec{u}_1)$ holds if there exists a deterministic PTIME program $\mathcal{S}$ computing $\vec{u}_i$ when given $\vec{v}_i$ as input, for both $i \in \{0, 1\}$. The connection is quite clear: if $(\emptyset, \emptyset) \triangleright (\vec{u}_0, \vec{u}_1)$ holds then $\vec{u}_0 \sim \vec{u}_1$ is a valid CCSA formula. Unfortunately, the simulator $\mathcal{S}$ that can be obtained using the approach of [6] is very limited: first, no interaction (through oracle calls) with an external game is supported; and second, the simulator $\mathcal{S}$ is always a simple *loop-free* (no while or for loops, no recursion) and *deterministic* program. This restricts the judgements that can be derived, as more complex bi-deduction properties require more involved simulators. In particular, the simulators constructed in the manual proofs of soundness of Squirrel cryptographic axioms [5, 8] are all completely out-of-scope.

Contributions. In this paper, we propose a formal framework for systematically deriving cryptographic axioms in the higher-order CCSA logic of Squirrel. To that end:

i) We significantly adapt the notion of bi-deduction introduced in [6] by enriching the computational capabilities of the simulator $\mathcal{S}$ witnessing a bi-deduction judgement:

- We allow *probabilistic* PTIME programs. This requires to carefully track — using *symbolic constraints* — which names correspond to adversarial computations and which belong to the game and, more generally, to carefully relate — through a *probabilistic coupling* — the random samplings performed in the logic and in the computations involving the game.
- We provide access to *oracles* of a cryptographic game G. Our approach supports games with an internal persistent state, whose properties can be tracked by extending the bi-deduction judgements with Hoare pre- and post-conditions. For instance, in the case of the PRF game, the state is used to keep track of which messages have been hashed, and the assertion that some message has not been hashed before will resurface in a conditional surrounding calls to the challenge oracle, corresponding to the ad hoc condition over elements of H in Example 1.

ii) We formally show that our improved notion of bi-deducibility is expressive enough to derive sound cryptographic axioms in our logic, and we provide a *proof system* for establishing bi-deducibility. Our proof system notably features an induction rule that can deal with the recursively defined observables of protocols.

iii) We design a heuristic proof-search algorithm for establishing bi-deduction in a fully automated (albeit incomplete) manner, which is capable of finding complex proofs by induction. We implement this algorithm in Squirrel, and validate our approach on several case studies.

Outline. We provide an overview of our approach in Section 2. Section 3 presents the needed background on Squirrel's higher-order CCSA logic, and on cryptographic games and adversaries. We develop in Section 4 our central contributions: bi-deduction w.r.t. a cryptographic game and a rich proof system for deriving bi-deducibility. Section 5 presents a proof-search procedure for this system. We report in Section 6 on some case studies performed with an extension of Squirrel implementing this procedure. Finally, we discuss related works in Section 7. Our extension and case studies have been merged into the main branch of the tool [27].

2 OVERVIEW

We shall start with a high-level description of the key notions in our work, building on an example security property for a simple

protocol. The Hash Lock protocol relies on on a keyed hash function $\mathsf{h}(\_,\_)$, and involves participants $T_1, T_2, \ldots$ where each T_i owns a secret hashing key k_i to be used across an unbounded number of sessions. For its j^{th} session, participant T_i inputs x and outputs $\langle \mathsf{n}_{i,j}, \mathsf{h}(\langle \mathsf{n}_{i,j}, x\rangle, \mathsf{k}_i)\rangle$, where $\langle \mathsf{n}_{i,j}, x\rangle$ is a pair combining a session-specific nonce, i.e. a fresh random sampling, and the input x.

We model an execution of the protocol along an arbitrary execution trace, given as a finite sequence of *timestamps*. Each timestamp in the trace corresponds to an elementary interaction between the adversary and some participant, where some T_i inputs a message from the adversary and outputs its answer. The timestamp where participant T_i plays its session j will be represented by $\mathsf{T}(i, j)$. All timestamps must be of this form, except for the initial timestamp, noted init, and the special value undef used to represent timestamps not present in the trace. We let pred be the predecessor function on timestamps that are present in the trace. We finally model the execution using three mutually recursive functions: $\mathsf{input}(t)$ represents the input provided to the protocol at time t; $\mathsf{output}(t)$ the protocol output at that time; and $\mathsf{frame}(t)$ the sequence of all outputs up to time t, included. As usual, we consider an active adversary that fully controls the network: it can read, intercept and even modify all messages exchanged by honest participants. The inputs of the protocol are then always the result of an adversarial computation, which we represent with the function att(_). Our functions can then be defined as follows:

$$\begin{aligned}
\mathsf{frame}(\mathsf{init}) = \mathsf{input}(\mathsf{init}) &= \mathsf{output}(\mathsf{init}) = \mathsf{empty}\\
\mathsf{frame}(\mathsf{t}) &= \langle \mathsf{frame}(\mathsf{pred}(\mathsf{t})), \mathsf{output}(\mathsf{t})\rangle\\
\mathsf{input}(\mathsf{t}) &= \mathsf{att}(\mathsf{frame}(\mathsf{pred}(\mathsf{t}))) \qquad (\text{when } \mathsf{init} < \mathsf{t})\\
\mathsf{output}(\mathsf{T(i,j)}) &= \langle \mathsf{n}_{i,j}, \mathsf{h}(\langle \mathsf{n}_{i,j}, \mathsf{input}(\mathsf{T(i,j)})\rangle, \mathsf{k}_i)\rangle
\end{aligned}$$

Using these functions, it is then possible to express security properties of the protocol as logical formulas. We are interested in proving that Hash Lock ensures a form of key secrecy: *when outputting the hash keyed by key k the protocol doesn't reveal any information about k to the adversary*. More formally, we would like to show that, at any point of an interaction of the adversary with the protocol, the adversary cannot distinguish a new hash $\mathsf{h}(\langle \mathsf{n}_{i_0,j_0}, \mathsf{input}(\mathsf{T}(i_0, j_0))\rangle, \mathsf{k}_{i_0})$ from a randomly sampled value. In the CCSA approach, the adversary interacting with the protocol and attempting to distinguish two scenarios is split in several parts: the computation of the inputs represented by att(_) on the one hand, and the implicit distinguisher in the computational indistinguishability predicate $\sim$ on the other. We thus encode our property by explicitly passing the knowledge obtained through the protocol execution (i.e., the frame) to the distinguisher. We seek to verify, for an arbitrary $t_0 = \mathsf{T}(i_0, j_0)$ and for a fresh name $\mathsf{n}_{\mathsf{fresh}}$:

$$\mathsf{frame}(\mathsf{pred}(t_0)), \mathsf{output}(t_0) \sim \mathsf{frame}(\mathsf{pred}(t_0)), \langle \mathsf{n}_{i_0,j_0}, \mathsf{n}_{\mathsf{fresh}}\rangle \quad (1)$$

To prove Eq. (1), we need a pseudo-randomness assumption on h: we rely on the PRF assumption of Example 1, which we define more precisely using the games of Fig. 1. Function h is said to be a PRF when the *advantage* of any PPTM in distinguishing G_0 and G_1 is negligible, i.e. for any PPTM $\mathcal{A}$, the probability

$$\left| Pr(\mathcal{A}^{\mathsf{G}_0}() = 1) - Pr(\mathcal{A}^{\mathsf{G}_1}() = 1) \right|$$

Initialization (G_0 *and* G_1)*:*

$$k \xleftarrow{\$};\ \ell_{\mathbf{hash}} \leftarrow [\,];\ \ell_{\mathbf{challenge}} \leftarrow [\,];$$

Hash and challenge oracles for game G_b ($\mathsf{b} \in \{0;1\}$)*:*

$$\begin{aligned}
&\textbf{oracle } \mathbf{hash}(x) := \{\ \ell_{\mathbf{hash}} \leftarrow x :: \ell_{\mathbf{hash}};\\
&\qquad \textbf{return } (\text{if } x \notin \ell_{\mathbf{challenge}} \text{ then } \mathsf{h}(x,k) \text{ else zero})\ \}\\
&\textbf{oracle } \mathbf{challenge}(x) := \{\ r \xleftarrow{\$};\\
&\qquad v \leftarrow \text{if } x \notin \ell_{\mathbf{hash}} \cup \ell_{\mathbf{challenge}} \text{ then}\\
&\qquad\quad \text{if b then } \mathsf{h}(x,k) \text{ else } r\\
&\qquad \text{else zero ;}\\
&\qquad \ell_{\mathbf{challenge}} \leftarrow x :: \ell_{\mathbf{challenge}};\\
&\qquad \textbf{return } v\ \}
\end{aligned}$$

Remark: Queries to both oracles are logged in the lists $\ell_{\mathbf{hash}}$ and $\ell_{\mathbf{challenge}}$ to avoid repeated queries that would make the assumption trivially unfeasible.

Figure 1: Games for the PRF cryptographic assumption.

is asymptotically smaller than any η^{-k} for $k \geq 0$, where η is the *security parameter* — here, η is the size of the key k.

2.1 Cryptographic Reductions

We are going to prove our security property using a *cryptographic reduction* to the PRF game. More precisely, assuming a PTIME adversary $\mathcal{A}$ against the target indistinguishability of Eq. (1) we build a PTIME adversary $\mathcal{B}$ against the PRF game $(\mathsf{G}_0, \mathsf{G}_1)$ of Fig. 1 such that $\mathcal{B}$ is the composition $\mathcal{A} \circ \mathcal{S}$ of the adversary $\mathcal{A}$ with a *simulator* $\mathcal{S}$ computing the terms appearing on the left or right side of the security formula in Eq. (1) — depending on whether $\mathcal{S}$ has access to the oracles G_0 or G_1. Roughly, $\mathcal{S}$ satisfies:

$$\mathcal{S}^{\mathsf{G}_0}() = \left(\mathsf{frame}(\mathsf{pred}(t_0)), \mathsf{output}(t_0)\right)$$
$$\mathcal{S}^{\mathsf{G}_1}() = \left(\mathsf{frame}(\mathsf{pred}(t_0)), \langle \mathsf{n}_{i_0,j_0}, \mathsf{n}_{\mathsf{fresh}}\rangle\right)$$

Thus, $\mathcal{A}$'s advantage against Eq. (1) is exactly $\mathcal{B}$'s advantage against the PRF game $(\mathsf{G}_0, \mathsf{G}_1)$, which we assumed negligible.

The simulator $\mathcal{S}$ is described in a slightly beautified and simplified imperative language in Fig. 2. On lines 2–16, $\mathcal{S}$ computes the term $\mathsf{frame}(\mathsf{pred}(t_0))$.

Since frame is defined mutually recursively with input and output, $\mathcal{S}$ computes simultaneously all three functions for all timestamps in $\{\mathsf{init}; \ldots; \mathsf{pred}(t_0)\}$. Concretely, $\mathcal{S}$ uses three identically named arrays input, output, and frame indexed by timestamps, which are being filled by the **for** loop starting on line 2, following the recursive definition of input, frame and output.

In our setting, the simulator's and game's randomness is *early-sampled*: the simulator and game both have access to tagged random tapes from which they extract their random values; these tapes are implicitly sampled before the execution starts. On line 7, the simulator performs a random sampling $\mathsf{n} \xleftarrow{\$} \mathsf{T_S}[\mathsf{offset_n(i,j)}]$: this actually reads an early-sampled random tape at a position determined by the tag $\mathsf{T_S}$ (indicating a simulator sampling) and an offset associated to the name $\mathsf{n}_{i,j}$ being simulated.

Our notion of oracle call is also adapted to fit with the logic. On line 9 the simulator computes $\mathsf{h}(\langle \mathsf{n}_{i_0,j}, \mathsf{input}(\mathsf{T}(i_0, j_0))\rangle, \mathsf{k}_{i_0})$ using the oracle call $\mathsf{G}.\mathbf{hash}(\langle \mathsf{n}_{i_0,j}, \mathsf{input}[\,\mathsf{T}(i_0, j_0)\,]\rangle)[\mathsf{offset_k}(\mathsf{i}_0)]$. In addition to passing the message to be hashed as an argument, the

```
1  (* Recursively compute input, output and frame using the hash oracle. *)
2  for each t ∈ [ init; t0 [ {
3    match t with
4    | init → input[t] ← empty; output[t] ← empty; frame[t] ← empty
5    | T(i,j) →
6      input[t] ← att(frame[pred t])
7      n ←$ TS[offsetn(i, j)]   (* pre-sampled value access *)
8      if i = i0 then {
9        x ← G.hash(⟨ n, input[t] ⟩)[offsetk(i0)]   (* oracle call *)
10     } else {
11       k ←$ TS[offsetk(i)]   (* pre-sampled value access *)
12       x ← h(⟨ n, input[t] ⟩, k)
13     }
14     output[t] ← ⟨ n, x ⟩
15     frame[t] ← ⟨ frame[pred t], output[t] ⟩
16 }
17
18 (* Use the challenge oracle to compute output(t0) or ⟨n_{i0,j0}, n_fresh⟩. *)
19 input[t0] ← att(frame[pred t0])
20 n ←$ TS[offsetn(i0, j0)]
21 output' ← ⟨ n, G.challenge(⟨ n, input[t0] ⟩)[offsetk(i0), offset_{n_fresh}()] ⟩
22 return (frame[pred t0], output')
```

Figure 2: Reduction to the PRF assumption.

simulator specifies here where the oracle should read the key: although this value is usually understood as being sampled when the game initializes, this is irrelevant in our early-sampled semantics; of course, our model forbids that the simulator calls oracles with inconsistent values for the key's offset, and the simulator cannot read the random tape at this position. This unusual setup will be useful, again, to help track the relationship between the samplings and the names that are being simulated.

On line 18, $\mathsf{frame}[\mathsf{pred}(t_0)]$ has been properly computed and can be used to compute in output' a value which will be $\mathsf{output}(t_0)$ on the left and $\langle \mathsf{n}_{i_0,j_0}, \mathsf{n}_{\mathsf{fresh}} \rangle$ on the right. This is done using the challenge oracle G.**challenge**, passing the offsets for the key and the fresh sampling. Crucially, the call to the challenge oracle returns the expected value because the input $\langle \mathsf{n}_{i_0,j_0}, \mathsf{input}(t_0) \rangle$ has never been queried to the hash oracle, except with negligible probability. Indeed, the hash oracle G.**hash** has only been queried on the values:

$$\mathcal{H} \stackrel{\text{def}}{=} \left\{ \langle \mathsf{n}_{\mathsf{i,j}}, \mathsf{input}(t) \rangle \mid t = \mathsf{T}(\mathsf{i},\mathsf{j}) < t_0 \right\}$$

and the probability that a collision occurs between $\langle \mathsf{n}_{i_0,j_0}, \mathsf{input}(t_0) \rangle$ and a value in $\mathcal{H}$ is bounded by:

$$|\mathcal{H}| \times \Pr(\mathsf{n}_{i_0,j_0} = \mathsf{n}_{i,j}) \qquad (\text{for } (\mathsf{i},\mathsf{j}) \neq (\mathsf{i}_0,\mathsf{j}_0))$$

This is negligible since $\mathcal{H}$ is a set of constant size w.r.t. η, and since the names n_{i_0,j_0} and $\mathsf{n}_{i,j}$ are independent uniform random samplings in an exponentially large set.

2.2 Cryptographic Reductions by Bi-deduction

We now informally describe how our approach allows to infer such a simulator by significantly extending the notion of *bi-deduction* introduced in [6]. First, let us present what is (mono-)deduction: we say that terms $\vec{v}$ can be deduced from terms $\vec{u}$ if there exists a polynomial-time simulator $\mathcal{S}$ such that $\mathcal{S}(\vec{u}) = \vec{v}$. Optionally, $\mathcal{S}$ may be given access to the oracles of a game G, in which case we must have $\mathcal{S}^{\mathsf{G}}(\vec{u}) = \vec{v}$.

Bi-deduction. In bi-deduction, the initial knowledge $\vec{u}$ and the target $\vec{v}$ are replaced by pairs of vectors of terms, respectively $\#(\vec{u}_0;\vec{u}_1)$ and $\#(\vec{v}_0;\vec{v}_1)$, called *bi-terms*, which typically represent messages in the left and right scenarios of an indistinguishability. We use a dash # to distinguish pairing of the left and right scenarios from the standard pairing that can appear in the terms $\vec{u}_0, \vec{u}_1, \vec{v}_0, \vec{v}_1$. For example, the indistinguishability in Eq. (1) can be represented by

$$\#(\mathsf{frame}(\mathsf{pred}(t_0)), \mathsf{output}(t_0);\quad \mathsf{frame}(\mathsf{pred}(t_0)), \langle \mathsf{n}_{i_0,j_0}, \mathsf{n}_{\mathsf{fresh}} \rangle)$$

or, alternatively,

$$\mathsf{frame}(\mathsf{pred}(t_0)), \#(\mathsf{output}(t_0); \langle \mathsf{n}_{i_0,j_0}, \mathsf{n}_{\mathsf{fresh}} \rangle)$$

by factorizing common parts of the left and right terms. Informally, we say that $\#(\vec{u}_0;\vec{u}_1)$ bi-deduces $\#(\vec{v}_0;\vec{v}_1)$ with access to the pair of games $(\mathrm{G}_0, \mathrm{G}_1)$, which we write

$$\#(\vec{u}_0;\vec{u}_1) \rhd_{(\mathrm{G}_0,\mathrm{G}_1)} \#(\vec{v}_0;\vec{v}_1)$$

if there exists a *single* polynomial-time simulator $\mathcal{S}$ such that $\mathcal{S}^{\mathrm{G}_0}(\vec{u}_0) = \vec{v}_0$ and $\mathcal{S}^{\mathrm{G}_1}(\vec{u}_1) = \vec{v}_1$. The cryptographic reduction argument of Section 2.1 is captured by the rule:

$$\frac{\emptyset \rhd_{(\mathrm{G}_0,\mathrm{G}_1)} \#(\vec{v}_0;\vec{v}_1)}{\vec{v}_0 \sim \vec{v}_1}\ \textsc{Bi-deduce} \qquad (2)$$

To prove the soundness of this rule, assume that its conclusion does not hold: then the adversary $\mathcal{A}$ against $\vec{v}_0 \sim \vec{v}_1$ can be composed with the simulator $\mathcal{S}$ witnessing $\emptyset \rhd_{(\mathrm{G}_0,\mathrm{G}_1)} \#(\vec{v}_0;\vec{v}_1)$ to obtain a adversary $\mathcal{B} = \mathcal{A} \circ \mathcal{S}$ against the games $(\mathrm{G}_0, \mathrm{G}_1)$. Thus, $\mathcal{S}$ is a sub-procedure of the adversary $\mathcal{B}$ against the cryptographic game $(\mathrm{G}_0, \mathrm{G}_1)$ we are reducing to — this is why it is crucial that the same simulator $\mathcal{S}$ is used for both side of the bi-deduction judgement. Note that this rule requires that the input of the bi-deduction premise is empty: non-empty inputs $\#(\vec{u}_0;\vec{u}_1)$ will be useful later, e.g. to support transitivity and inductive reasoning steps in our proof system for bi-deduction.

Proof system. We design a proof system for our extended notion of bi-deduction, which we use as justification of a syntax-directed proof-search procedure. This procedure is heuristic, and can be used to automatically derive bi-deduction judgements and through them build simulators like the one described in Fig. 2. The bi-deduction rules of our proof-system allow to build simulators piece-by-piece in a compositional way, using the simulators provided by the premises of a rule as sub-procedures of the simulator being built to justify the rule's conclusion. We describe next a few *simplified* and *informal* rules of our proof system, and show how they can be used to automatically infer simulators. In the rules presented below, we omit the games and write $\rhd$ instead of $\rhd_{(\mathrm{G}_0,\mathrm{G}_1)}$. Also, bi-terms will be annotated by a sharp # in sub-script, e.g. $\vec{u}_\#$, $\vec{v}_\#$ and $\vec{w}_\#$ are all bi-terms. Finally, rules come with an associated (informal) program showing how the simulators provided by the premises are composed to obtain the simulator for the conclusion.

The *transitivity rule* allows to decompose a bi-deduction $\vec{u}_\# \rhd \vec{v}_\#, \vec{w}_\#$ into a bi-deduction of $\vec{v}_\#$ followed by a bi-deduction of $\vec{w}_\#$

with $\vec{v}_\#$ as additional input:

$$\frac{\vec{u}_\# \rhd \vec{v}_\# \qquad \vec{u}_\#, \vec{v}_\# \rhd \vec{w}_\#}{\vec{u}_\# \rhd \vec{v}_\#, \vec{w}_\#} \qquad \underline{\mathcal{S}(\vec{u})} := \begin{array}{l} \vec{v} \leftarrow \mathcal{S}_1(\vec{u}); \\ \vec{w} \leftarrow \mathcal{S}_2(\vec{u}, \vec{v}) \end{array}$$

Coming back to Fig. 2, this rule allows to combine the simulator $\mathcal{S}_1$ (lines 2–16) of $\emptyset \rhd \mathsf{frame}(\mathsf{pred}(t_0))$ with the simulator $\mathcal{S}_2$ (lines 19–21) of $\mathsf{frame}(\mathsf{pred}(t_0)) \rhd \#(\mathsf{output}(t_0); \langle \mathsf{n}_{i_0,j_0}, \mathsf{n}_{\mathsf{fresh}} \rangle)$ to obtain the full simulator of Fig. 2 which proves the equivalence in Eq. (1).

The *name rule* lets the simulator sample a random value directly:

$$\frac{}{\vec{u}_\# \rhd \mathsf{n}_i} \qquad \underline{\mathcal{S}(\vec{u})} := \mathsf{x}_\mathsf{n} \xleftarrow{\$} \mathsf{T}_\mathsf{S}[\mathsf{offset}_\mathsf{n}(\mathsf{i})]$$

For example, this rule allows to infer the three simulators line 7, line 11 and line 20 in Fig. 2.

The *oracle rule* allows the simulator to call any oracle of the game $\mathsf{G} \in \{\mathsf{G}_0, \mathsf{G}_1\}$ on any input it can compute. For instance, in the case of the hash oracle of the PRF game of Fig. 1:

$$\frac{\vec{u}_\# \rhd \vec{v}_\#}{\vec{u}_\# \rhd \mathsf{h}(\vec{v}_\#, \mathsf{k}_\mathsf{i})} \qquad \underline{\mathcal{S}(\vec{u})} := \begin{array}{l} \vec{v} \leftarrow \mathcal{S}_1(\vec{u}); \\ \mathsf{x}_\mathsf{h} \leftarrow \mathsf{G}.\mathbf{hash}(\vec{v})[\mathsf{offset}_\mathsf{k}(\mathsf{i})] \end{array}$$

This is the rule justifying the oracle call on line 9 of Fig. 2.

Randomness usage. The addition of these rules creates two issues:

i) We should not be able to use the oracle rule twice on two different keys k_i and k_j, since the PRF game only supports hashing with a single and fixed key.
ii) We should not be allowed to sample a name in the simulator (through the name rule) and use this name to represent a sampling in the game (through the oracle rule).

We solve these issues by equipping our bi-deduction judgement with a system of *name constraints* that allow to track the usage of randomness by the simulator. Our constraint systems notably allow us to express: *consistency* conditions on the secret keys of the game, ensuring that oracle calls are always asked to use the same offset for the same secret key; *ownership* of random samplings, preventing the simulator from directly accessing random values that are used as secret keys by the game. Concretely, constraint systems are recorded in bi-deduction judgements, and their validity is deferred until the end of the bi-deduction derivation. E.g. the name and bi-deduction rule Bi-deduce roughly look like this:

$$\frac{}{\{(\mathsf{n}_i, \mathsf{T}_\mathsf{S})\} \vdash \vec{u}_\# \rhd \mathsf{n}_i} \qquad \frac{\vdash \mathrm{Valid}(C) \qquad C; \emptyset \rhd \#(\vec{v}_0; \vec{v}_1)}{\vec{v}_0 \sim \vec{v}_1}$$

Here, $(\mathsf{n}_i, \mathsf{T}_\mathsf{S})$ records that n_i has been sampled by the simulator, and $\mathrm{Valid}(C)$ is a standard Squirrel formula — that can be discharged to the user — ensuring that the constraint system C is valid.

Stateful games. Recall that the oracles of the PRF game are guarded by conditions involving the logs $\ell_{\mathbf{hash}}$ and $\ell_{\mathbf{challenge}}$. E.g., the value $\langle \mathsf{n}_{\mathsf{i}_0,\mathsf{j}_0}, \mathsf{input}(\mathsf{T}(\mathsf{i}_0, \mathsf{j}_0)) \rangle$ sent to the challenge oracle at the end of the simulator (line 21 of Fig. 2) must not have been already queried to the hash oracle **hash** — except with negligible probability. As discussed in the previous section, proving this requires to establish a property on the game's internal memory, namely that the set of previously hash values is of the form:

$$\left\{ \langle \mathsf{n}_{\mathsf{i},\mathsf{j}}, \mathsf{input}(t) \rangle \mid t = \mathsf{T}(\mathsf{i}, \mathsf{j}) < \mathsf{T}(\mathsf{i}_0, \mathsf{j}_0) \right\}$$

To account for that, we equip our bi-deduction judgement with a Hoare-style pre-condition φ and post-condition ψ to track the memory state of the game. Putting everything together, we shall use bi-deduction judgements that are roughly of the form:

$$C, (\varphi, \psi) \vdash \#(\vec{u}_0; \vec{u}_1) \rhd_{(\mathsf{G}_0, \mathsf{G}_1)} \#(\vec{v}_0; \vec{v}_1)$$

Informally, this judgement states that there exists a simulator $\mathcal{S}$ using randomness as prescribed by the name constraints C such that, for any $i \in \{0; 1\}$, the execution of $\mathcal{S}$ on input $\vec{u}_i$ starting from a game G in the initial state satisfying φ computes $\vec{v}_i$ and leaves the game G in a final state satisfying ψ.

Discussion. The notion of bi-deduction we introduce in this paper supports simulators whose capacities go beyond what was possible using the basic bi-deduction of [6]. In particular, we consider simulators that are *probabilistic* programs with access to *stateful oracles*. As shown in the example presented above, supporting these features requires to extend bi-deduction in two non-trivial ways: we record the randomness usage of the simulator using name constraints (see Section 4.1) and we use Hoare-style pre- and post-condition to track the state of the game's internal memory (see Section 4.2). While the latter extension is standard in program logics, the former is a novel contribution of this paper.

3 DEFINITIONS

We now describe the essential parts of our formal setup: the higher-order CCSA logic (Section 3.1) and formal definitions of cryptographic games and adversaries (Section 3.2) with an early-sampling semantics and tailored samplings and oracle calls that will facilitate relating imperative programs and logical terms. We shall use colors to help distinguishing types, logical and computational notions.

3.1 Higher-Order CCSA Logic

We recall the main features of the indistinguishability logic of [8] (details can be found in Appendix A).This is a first-order logic over higher-order terms with a probabilistic semantics. More precisely, models of the logic interpret terms as random variables sharing the same sample space. The logic features predicates capturing specific properties of the random variable interpretations of terms: most notably, $t_1 \sim t_2$ holds when the advantage of any PPTM in distinguishing t_1 from t_2 is negligible.

Types. We assume a set of base types $\mathbb{B}$, e.g. bool, message, int. A *type*, denoted by τ, is either a base type $\tau_\mathsf{b} \in \mathbb{B}$ or $\tau_1 \to \tau_2$ where τ_1 and τ_2 are types. A type τ is interpreted in a *type structure* $\mathbb{M}$ as an η-indexed collections of sets $(\llbracket \tau \rrbracket^\eta_\mathbb{M})_{\eta \in \mathbb{N}}$, in such a way that $\llbracket \tau_1 \to \tau_2 \rrbracket^\eta_\mathbb{M} = \llbracket \tau_1 \rrbracket^\eta_\mathbb{M} \to \llbracket \tau_2 \rrbracket^\eta_\mathbb{M}$. We require that $\llbracket \mathsf{bool} \rrbracket^\eta_\mathbb{M} = \{0, 1\}$.

Terms. We consider simply-typed λ-terms, using variables in $\mathcal{X}$:

$$t ::= x \mid (t\ t) \mid \lambda(x : \tau).\, t \qquad (\text{when } x \in \mathcal{X})$$

A term is typed, according to standard rules, w.r.t. an *environment* $\mathcal{E}$, which is a sequence of: *declarations* of the form $(x : \tau)$, which states the existence of a variable x of type τ; and *definitions* of the form $(x : \tau = t)$, which introduces a new variable with a fixed meaning. Recursive definitions are allowed, subject to a well-foundedness condition [8]. We write $\mathcal{E} \vdash t : \tau$ when t has type τ in $\mathcal{E}$, and we only consider well-typed terms, modulo α-renaming.

Models. A *model* $\mathbb{M}$ for an environment $\mathcal{E}$, which we write $\mathbb{M} : \mathcal{E}$, extends an identically named type-structure $\mathbb{M}$ to interpret well-typed terms as random variables on the interpretation of their types. The model $\mathbb{M}$ first specifies, for every η, a sample space $\mathbb{T}_{\mathbb{M},\eta}$, whose elements are pairs $\rho = (\rho_a, \rho_h)$ of *random tapes* given by finite bitstrings: ρ_a will be used to model *adversarial* randomness while ρ_h will be used for *honest* random samplings. Given a type τ, $\mathbb{RV}_{\mathbb{M}}(\tau)$ is the set of η-indexed families of random variables from $\mathbb{T}_{\mathbb{M},\eta}$ to $[\![\tau]\!]^{\eta}_{\mathbb{M}}$:

$$\mathbb{RV}_{\mathbb{M}}(\tau) \stackrel{\text{def}}{=} \left\{(X_\eta)_{\eta\in\mathbb{N}} \mid X_\eta : \mathbb{T}_{\mathbb{M},\eta} \to [\![\tau]\!]^{\eta}_{\mathbb{M}} \text{ for every } \eta\right\}$$

For each variable $(x : \tau)$ introduced in $\mathcal{E}$, the model $\mathbb{M}$ provides an interpretation $\mathbb{M}(x) \in \mathbb{RV}_{\mathbb{M}}(\tau)$. This is then lifted naturally to interpret any term t of type τ in $\mathcal{E}$ into $[\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} \in \mathbb{RV}_{\mathbb{M}}(\tau)$. If $\mathcal{E}$ contains a (possibly recursive) definition $x : \tau = t$, we have $[\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = [\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$ for any model $\mathbb{M} : \mathcal{E}$. Details may be found in [8].

Builtins and local formulas. For convenience, we assume that environments declare some builtin symbols, and we restrict to models where they are interpreted as expected. Builtins include

$$\wedge, \vee, \Rightarrow : \mathsf{bool} \to \mathsf{bool} \to \mathsf{bool} \qquad \neg : \mathsf{bool} \to \mathsf{bool}$$
$$=_\tau : \tau \to \tau \to \mathsf{bool} \quad \forall_\tau, \exists_\tau : (\tau \to \mathsf{bool}) \to \mathsf{bool} \quad \text{(for each } \tau\text{)}$$

and their interpretation notably satisfies:

$$[\![=_\tau]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(a, a') = 1 \in [\![\mathsf{bool}]\!]^{\eta}_{\mathbb{M}} \text{ iff. } a = a' \qquad (a, a' \in [\![\tau]\!]^{\eta}_{\mathbb{M}})$$
$$[\![\forall_\tau]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(f) = 1 \in [\![\mathsf{bool}]\!]^{\eta}_{\mathbb{M}} \text{ iff. } f(a) = 1 \text{ for all } a \in [\![\tau]\!]^{\eta}_{\mathbb{M}}$$

We generally omit type subscripts and use standard infix notations: we may write, for example, $\forall x : \tau.\ f(x) \Rightarrow x = g(x)$ when $f : \tau \to \mathsf{bool}$ and $g : \tau \to \tau$. Hence terms of type bool can be seen as formulas, which we call *local formulas*. The semantics of a local formula is a family of boolean random variables in $\mathbb{RV}_{\mathbb{M}}(\mathsf{bool})$.

Names. A crucial ingredient in the CCSA approach is to have symbols representing random samplings. We let $\mathcal{N} \subseteq \mathcal{X}$ be a set of symbols called *names*. A name $\mathsf{n} \in \mathcal{N}$ can only be *declared* in an environment, and must have a type of the form $\tau_0 \to \tau$ where the *index* type τ_0 must be finite, which we write finite(τ_0): concretely, $[\![\tau_0]\!]^{\eta}_{\mathbb{M}}$ must be finite for all η. We restrict to models where names have a specific interpretation: all name instances $(\mathsf{n}_0\ i_0)$ of a given type τ must correspond to the same probability distribution; moreover, two distinct instances $(\mathsf{n}_0\ i_0)$ and $(\mathsf{n}_1\ i_1)$ (i.e. the name symbols are distinct or $i_0 \neq i_1$) must correspond to independent random variables. For convenience, we allow a name to have a base type: the semantical assumptions on it are then the same as if it were indexed over unit.

Example 2. *To model the Hash Lock protocol of Section 2, we would use names* $\mathsf{n} : \mathsf{index} \times \mathsf{index} \to \mathsf{message}$, $\mathsf{k} : \mathsf{index} \to \mathsf{key}$ *and* $\mathsf{n}_{\mathsf{fresh}} : \mathsf{message}$.

Global Formulas. The higher-order CCSA logic is a first-order logic over our higher-order terms. To avoid confusion with the local formulas, noted φ, the actual formulas of the logic are called *global* and noted F. We also distinguish connectives, e.g. global conjunction is noted $\tilde{\wedge}$. We introduce next some useful global predicates.

For any term t of base type, $\mathsf{adv}(t)$ states that the interpretation of t can be computed by a PPTM which can only access the adversarial random tape: $\mathbb{M} \models \mathsf{adv}(t)$ iff. there exists a PPTM $\mathcal{M}$ s.t. $\mathcal{M}(1^\eta, \rho_a) = [\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$ for all $\eta \in \mathbb{N}$ and $\rho = (\rho_a, \rho_h) \in \mathbb{T}_{\mathbb{M},\eta}$. The definition is extended naturally to terms t of type $\tau^1_b \to \ldots \to \tau^k_b \to \tau_b$ by requiring the existence of a PPTM taking arguments as inputs. E.g. $\mathsf{adv}(\mathsf{n}_{\mathsf{fresh}})$ never holds since names are sampled from the honest random tape, and we require that $\mathsf{adv}(\mathsf{att})$ holds in any model.

For any boolean term t, the atoms $[t]_e$ and $[t]$ state, respectively, that t is exactly true and *overwhelmingly* true:

$$\mathbb{M} \models [t]_e \quad \text{iff} \quad [\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = 1 \text{ for all } \eta \in \mathbb{N}, \rho \in \mathbb{T}_{\mathbb{M},\eta}$$
$$\mathbb{M} \models [t] \quad \text{iff} \quad \Pr_{\rho\in\mathbb{T}_{\mathbb{M},\eta}}([\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = 0) \text{ is negligible in } \eta$$

Finally, for $\vec{t} = t_1, \ldots, t_n$ and $\vec{t'} = t'_1, \ldots, t'_n$ such that t_i and t'_i have the same base type for every i, the atom $\vec{t} \sim \vec{t'}$ states that $\vec{t}$ and $\vec{t'}$ are computationally indistinguishable. More precisely, $\mathbb{M} \models \vec{t} \sim \vec{t'}$ holds when, for any PPTM $\mathcal{D}$, the following quantity is negligible in η:

$$\left|\Pr_{\rho\in\mathbb{T}_{\mathbb{M},\eta}}(\mathcal{D}(1^\eta, [\![\vec{t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, \rho_a) = 1) - \Pr_{\rho\in\mathbb{T}_{\mathbb{M},\eta}}(\mathcal{D}(1^\eta, [\![\vec{t'}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, \rho_a) = 1)\right|$$

Following standard notations from first-order logic, we write $\mathcal{E}, \Theta \models \Theta'$ if all models w.r.t. $\mathcal{E}$ of the formulas of Θ are also models of the formulas of Θ'. A formula F w.r.t. $\mathcal{E}$ is *valid* when $\mathcal{E}, \emptyset \models F$. We shall use sequents of the form $\mathcal{E}, \Theta \vdash F$. This sequent is valid when $\mathcal{E}, \Theta \models F$.

Example 3. *If t is a term such that any free variable x of t is a name other than* n *or such that* $\mathsf{adv}(x)$ *holds, then the random variables* $[\![t]\!]^{\eta}_{\mathbb{M}:\mathcal{E}}$ *and* $[\![\mathsf{n}]\!]^{\eta}_{\mathbb{M}:\mathcal{E}}$ *are independent. If we further assume that the sample space associated to* n *is large enough, then* $[t \neq \mathsf{n}]$ *is valid.*

3.2 Cryptographic Games

We now describe our formal definitions of cryptographic games and adversaries. As shown in Section 2, we use a minor variation on the standard notions from cryptography, in order to facilitate relating samplings in the logical and computational settings.

3.2.1 Syntax. We assume a set of program variables $\mathcal{X}_p$, and an intrinsic typing associating to each variable $v \in \mathcal{X}_p$ a base type — we do not need a higher-order programming language. The *library* of our language, denoted by $\mathcal{L}_p$, is a set of typed function symbols disjoint from $\mathcal{X}_p$ representing built-in functions shared with the logic — i.e. we will have $\mathcal{L}_p \subseteq \mathcal{E}$.

We form well-typed expressions from $\mathcal{X}_p$, $\mathcal{L}_p$, and a special constant b of type bool. This constant will denote the side we are in when describing a left or right cryptographic game. Formally:

$$e_1, \ldots, e_n \in \mathsf{Expr} ::= e_1\ e_2 \mid v \mid f \mid \mathsf{b} \qquad (v \in \mathcal{X}_p, f \in \mathcal{L}_p)$$

Games. Cryptographic games set up some data (e.g. randomly sample keys) and provide functionalities through *oracles* to compute over this data, possibly changing it. Computations performed by oracles are described by *simple programs* written in a standard deterministic While language. As explained before, we are interested in pairs of games $(\mathsf{G}_0, \mathsf{G}_1)$ that are assumed to be indistinguishable; such pairs will be described by a single game G using the special variable b, i.e. G_i is obtained from G when $\mathsf{b} = i$.

Definition 1. *A game* $\mathsf{G} = (\mathcal{O}, \mathsf{decls})$ *is a finite set of oracle names $\mathcal{O}$, and a sequence of declarations* decls *according to the syntax given in Fig. 3. Declarations contain, in order, sequences of:*

1) initialization of variables, either:

$$
\begin{array}{lll}
\text{decl\_var} & ::= v \leftarrow e & (v \in \mathcal{X}_{\mathsf{p}}, e \in \text{Expr}) \\
\text{decl\_sample} & ::= v \xleftarrow{\$} & (v \in \mathcal{X}_{\mathsf{p}}) \\
\text{decl\_oracle} & ::= \textbf{oracle}\ f(\vec{v}) := \{\text{decl\_sample}^*;\ \mathsf{p};\ \textbf{return}\ e\} & \\
 & \qquad (f \in O,\ \vec{v} \in \mathcal{X}_{\mathsf{p}}^*,\ e \in \text{Expr},\ \mathsf{p}\ \text{a program}) & \\
\text{decls} & ::= \text{decl\_sample}^*;\ \text{decl\_var}^*;\ \text{decl\_oracle}^* &
\end{array}
$$

Figure 3: Syntax of games defined over oracle names O.

$$
\begin{array}{lll}
\mathsf{p}_1, \ldots, \mathsf{p}_n ::= & v \leftarrow e & \mid \textbf{skip} \\
 & \mid v \xleftarrow{\$} \mathsf{T}[e] & \mid \mathsf{p}_1; \mathsf{p}_2 \\
 & \mid v \leftarrow O_f(\vec{e})[\vec{e_g}; \vec{e_l}] & \mid \textbf{if}\ e\ \textbf{then}\ \mathsf{p}_1\ \textbf{else}\ \mathsf{p}_2 \\
 & \mid \textbf{abort} & \mid \textbf{while}\ e\ \textbf{do}\ \mathsf{p}
\end{array}
$$

Figure 4: Syntax of programs.

- $v_g \xleftarrow{\$}$, *which initializes the* global *random variable* v_g*;*
- *or* $v \leftarrow e$*, which assigns the evaluation of expression* e *to* v*.*

2) **oracle** $f(\vec{v}) := \{v_1^l \xleftarrow{\$}; \ldots; v_k^l \xleftarrow{\$};\ \mathsf{p};\ \text{return}\ e\}$*, which defines an oracle* $f \in O$ *with inputs* $\vec{v}$ *that initializes a sequence of* local *random variables* $v_1^l, \ldots, v_k^l$*, then executes a simple program* p *and finally returns* e*. We assume that* p *never modifies the values of the random global variables (e.g.* v_g *above), and the random local variables of any oracle (e.g.* $v_1^l, \ldots, v_k^l$ *above).*

We require that a game provides a single definition for each oracle name $f \in O$*. Given one such definition, we let:*

$$f.\text{args} \stackrel{\text{def}}{=} \vec{v} \quad f.\text{loc}_\$ \stackrel{\text{def}}{=} (v_1^l, \ldots, v_k^l) \quad f.\text{prog} \stackrel{\text{def}}{=} \mathsf{p} \quad f.\text{expr} \stackrel{\text{def}}{=} e$$

We also let $f.\text{glob}_\$$ *be the vector of all global random variables that are used in the oracle* f*.*

The PRF game shown in Fig. 1 is an instance of this notion of game. Note that the restriction to simple programs (which cannot perform random samplings nor oracle calls) in the body of oracles is without loss of generality.

For the rest of the article, we fix an arbitrary game G.

Adversaries. To model game adversaries, we extend simple programs with random samplings and oracle calls. As illustrated with the simulator of Fig. 2, we follow a style that fits well with our logic.

To this end, we use *tagged random samplings* from eagerly sampled tapes, where each tag in $\mathsf{Tag} = \{\mathsf{T_A}, \mathsf{T_S}, \mathsf{T_G}\}$ represents a source of randomness used during the execution of the simulator $\mathcal{S}$:

- $\mathsf{T_A}$ is for random samplings performed by $\mathcal{S}$ that will correspond to samplings in ρ_{a};
- $\mathsf{T_S}$ is for random samplings by $\mathcal{S}$ that will correspond to samplings in ρ_{h};
- $\mathsf{T_G}$ is for the game (oracle) randomness that $\mathcal{S}$ cannot directly access, and which will correspond to samplings in ρ_{h}.

Our syntax for programs, given in Fig. 4, extends the standard While language constructs with random samplings and oracle calls. The instruction $v \xleftarrow{\$} \mathsf{T}[e]$, where $v : \tau \in \mathcal{X}_{\mathsf{p}}$, $\mathsf{T} \in \mathsf{Tag}$, and e is of type int, samples a value of type τ using the randomness from random source T read at offset e, and stores it into v. The instruction $v \leftarrow O_f(\vec{e})[\vec{e_g}; \vec{e_l}]$ is an oracle call, where the variable v receives the call's result, f is the oracle being called, $\vec{e}$ are the oracle inputs, and $\vec{e_g}, \vec{e_l}$ have type int. These integers let the adversary control the offsets at which the oracle reads its randomness — the adversary controls the randomness offsets, but cannot read nor write these random bits itself. We distinguish the *global* offsets $\vec{e_g}$ used for the global random variables of the game from the *local* offsets $\vec{e_l}$ used for the local random variables of the oracles. As each oracle call must use fresh randomness for local samplings, our semantics will forbid the adversary from re-using local integer offsets. Similarly, global offsets will have to be consistent from one call to the next, as the game's global variables must be sampled only once.

3.2.2 Semantics. The semantics of a program interacting with a game will be parameterized by a model $\mathbb{M}$ of $\mathcal{L}_{\mathsf{p}}$ specifying the semantics of types and library functions, the values of the security parameter and the side b, a program random tape, and an initial memory. We only provide below the key elements of its definition; full details may be found in Appendix B.

A *memory* $\mu \in \mathsf{Mem}_{\mathbb{M},\eta}$ w.r.t. a type structure $\mathbb{M}$ and η is a function that associates to any variable $v \in \mathcal{X}_{\mathsf{p}}$ of type τ a value $\mu(v) \in [\![\tau]\!]^{\eta}_{\mathbb{M}}$. As usual, $\mu[v \mapsto a]$ is the memory such that $(\mu[v \mapsto a])(v) = a$ and $(\mu[v \mapsto a])(v') = \mu(v')$ for any variable $v' \neq v$.

A *program random tape* $\mathfrak{p} \in \mathfrak{P}$ is a set of infinite bitstrings $\mathfrak{p}[T, \tau]$, one for each $T \in \mathsf{Tag}$ and type τ, each one structured in fixed-size blocks used to sample values in τ. We note $\mathfrak{p}|^{\eta,\mathbb{M}}_{T,\tau}[k]$ the k^{th} such block.

Finally, the evaluation $(\!|\mathsf{p}|\!)^{\eta,\mathfrak{p}}_{\mathrm{G},\mathbb{M},i,\mu} \in \mathsf{Mem}_{\mathbb{M},\eta} \cup \{\bot\}$ of a program p against G_i (i.e. when $\mathsf{b} = i$) in memory μ and using the program random tape $\mathfrak{p}$ is either the memory obtained by executing p, or $\bot$ if the execution does not terminate. For the sake of conciseness, we write $(\!|\mathsf{p}|\!)^{\eta,\mathfrak{p}}_{\mu}$ when $\mathbb{M}$, i and G are clear from context.

3.2.3 Adversaries and Security. To define the security of a cryptographic game, we need to define which programs constitute valid *adversaries*. We require adversaries do not read nor write the game's internal variables nor the special side constant b, do not access tapes tagged $\mathsf{T_G}$, and properly use randomness offsets when invoking oracles: local offsets must be fresh, and global offsets must be consistent across oracle calls. More details are given in Appendix B.6.

Definition 2. *We use a special variable* res *to store the return value of a program. A game* G *is secure in a compatible model* $\mathbb{M}$ *if for any PTIME adversary* p*, the following quantity is negligible in* η*:*

$$\left|\Pr_{\mathfrak{p}}\left[(\!|\mathsf{p}|\!)^{\eta,\mathfrak{p}}_{\mathrm{G},\mathbb{M},0,\mu_0}[res] = 1\right) - \Pr_{\mathfrak{p}}\left[(\!|\mathsf{p}|\!)^{\eta,\mathfrak{p}}_{\mathrm{G},\mathbb{M},1,\mu_1}[res] = 1\right)\right|$$

where $\mu_i = \mu^{i\ \eta,\mathfrak{p}}_{init\ \mathbb{M}}$ *for any* $i \in \{0,1\}$ *is the initial memory of the game (seeAppendix B.3 for details).*

4 BI-DEDUCTION

We now develop our central concept: bi-deduction in presence of a cryptographic game. We will deal with several pairs of objects, where each component is involved in the deduction on one side $i \in \{0,1\}$ of the games. We introduce special notations for such pairs, following the style of [18].

Definition 3 (Bi-objects, $u_\#$, $\#(\_;\_)$)**.** *We call* bi-term *a pair of terms* $u_\# = \#(u_0; u_1)$*. We will similarly define and manipulate several kinds of bi-objects: for instance, we call (local) bi-formula a pair of*

local formulas $f_\# = \#(f_0, f_1;)$. We allow ourselves to factorize common parts of a bi-term (or any bi-object) by pushing the # downwards: e.g. $f(\#(u;v), g(\#(s;t)))$ denotes $\#(f(u, g(s)); f(v, g(t)))$.

We shall follow the intuitions given in Section 2, and derive a formal definition of bi-deduction for which we can prove that bi-deducibility entails indistinguishability. We start by introducing two necessary preliminary ingredients: constraints on the usage of random tapes, and assertions for describing the game's memory at a point of the simulator's computation.

4.1 Name Constraints

Our simulators can perform random samplings, either directly or indirectly through oracle calls. Names in the bi-deduction judgement will be used in both cases to represent such computations. For example, in the PRF game using key k, a simulator may compute $\mathsf{h}(m, \mathsf{k})$ through an oracle call (assuming that m is computable) but it may also compute $\mathsf{h}(m, \mathsf{s})$ explicitly when s and k are distinct names (by drawing s and computing the application of h itself). The two situations must be distinguished, as a simulator is forbidden from accessing the game's random samplings directly.

We introduce *name constraints* to keep track of how names are used in bi-deduction. We will make use of the following set of tags:

$$\mathsf{TAG}_{\mathsf{constr}} = \{\mathsf{T}_S, \mathsf{T}^{\mathsf{loc}}_{\mathsf{G}}\} \cup \{\mathsf{T}^{\mathsf{glob}}_{\mathsf{G},v} \mid v \in \mathsf{G.gs}\}$$

Tag T_S indicates that a name corresponds to a random sampling of the simulator; $\mathsf{T}^{\mathsf{loc}}_{\mathsf{G}}$ corresponds to an oracle's local sampling; finally, $\mathsf{T}^{\mathsf{glob}}_{\mathsf{G},v}$ corresponds to the global sampling of variable v in the game.

Definition 4. *A* name constraint *is a tuple $c = (\vec{\alpha}, \mathsf{n}, t, T, f)$ where $\vec{\alpha}$ are variables in $\mathcal{X}$ whose types are tagged* finite, n *is a name, t is a term, $T \in \mathsf{TAG}_{constr}$, and f is a local formula. A* constraint system *C is a list of name constraints.*

Intuitively, a constraint expresses that, for any arbitrary instantiation of the variables $\vec{\alpha}$ such that f holds, the name n is used at index t as specified by tag T. Variables $\vec{\alpha}$ are bound in the constraint. Accordingly, constraints are considered modulo renaming of these variables and, when we consider several constraints jointly, we implicitly assume that their bound variables are disjoint. We do *not* require that free variables of t and f are all bound by $\vec{\alpha}$.

Formally, we define the multiset $\mathcal{N}^{\eta,\rho}_{C,\mathbb{M}} \stackrel{\text{def}}{=} \bigcup_{c \in C} \mathcal{N}^{\eta,\rho}_{c,\mathbb{M}}$ where:

$$\mathcal{N}^{\eta,\rho}_{(\vec{\alpha},\mathsf{n},t,T,f),\mathbb{M}} \stackrel{\text{def}}{=} \{\langle \mathsf{n}, [\![t]\!]^{\eta,\rho}_{\mathbb{M}\sigma}, T\rangle \mid \mathsf{dom}(\sigma) = \vec{\alpha},\ [\![f]\!]^{\eta,\rho}_{\mathbb{M}\sigma} = \mathsf{true}\}$$

This interpretation of constraint systems supports a natural notion of constraint subsumption: we write $\mathcal{E}, \Theta \models C \subseteq C'$ when for any $\mathbb{M}$ such that $\mathbb{M} : \mathcal{E} \models \Theta$, for any η and ρ, we have the multiset inclusion $\mathcal{N}^{\eta,\rho}_{C,\mathbb{M}} \subseteq \mathcal{N}^{\eta,\rho}_{C',\mathbb{M}}$.

Example 4. *The system $[(\{i\}, \mathsf{n}, i, \mathsf{T}^{\mathsf{glob}}_{\mathsf{G},v}, f),\ (\{i\}, \mathsf{n}, i, \mathsf{T}_S, f')]$ expresses that: for every value of i for which f holds, (n i) represents the global sampling of the variable v of the game; and for every value of i for which f' holds, (n i) represents a sampling performed by the simulator. For this to make sense, we expect the formulas f and f' to be mutually exclusive, i.e. $[\forall i. \neg(f \wedge f')]_{\mathsf{e}}$ should be valid. Otherwise, there would exists a valuation v of i such that the index v of the name* n *would be tagged both as a simulator's and a game's sampling, which cannot happen in a valid interaction between the simulator and the game.*

We define a validity criterion for constraint systems that captures when the usage of names is consistent. First, name-tag associations must be *functional*: no name is associated to two different tags. Second, the local samplings must be *fresh*: the associated names do not occur anywhere else. Third, a globally sampled variable must be associated to a *unique* name. These three conditions must hold whenever the conditions f hold, and are defined formally as local formulas in Fig. 5. We finally define the *validity* of a constraint system C as the exact truth of all conditions on all pairs of constraint occurrences:

$$\mathrm{Valid}(C) \stackrel{\text{def}}{=} [\ \bigwedge_{c_1,c_2 \in C} \mathrm{Fun}(c_1,c_2) \wedge \mathrm{Fresh}(c_1,c_2) \wedge \mathrm{Unique}(c_1,c_2)\ ]_{\mathsf{e}}$$

As expected, $\Theta \models \mathrm{Valid}(C')$ and $\Theta \models C \subseteq C'$ imply $\Theta \models \mathrm{Valid}(C)$. The validity condition relies on the exact truth predicate, in other words we require our simulator to always behave correctly w.r.t. randomness usage. Importantly, we never require that names are distinct but only that their indices are distinct. The former would be too strong: we certainly do not rule out the possibility that a simulator, performing a random sampling by itself, happens to obtain the same value as a game's random sampling.

We will make use of bi-systems of constraints $C_\#$. In practice, they will be pairs of lists of the same length, so we view them as lists of bi-constraints. We define $\mathrm{Valid}(C_\#)$ as $\mathrm{Valid}(C_1) \mathbin{\tilde{\wedge}} \mathrm{Valid}(C_2)$.

4.2 Assertion Logic

As explained in Section 2, we need to keep track of the game's memory during the simulator's computation. We shall thus equip our bi-deduction judgement with pre- and post-conditions, relying on an abstract assertion language — a possible concrete instance of it will be taken in our implementation.

We thus assume an arbitrary language of assertions, with a notion of well-typedness w.r.t. environments, and a notion of satisfaction: given some environment $\mathcal{E}$, an assertion φ that is well-typed w.r.t. $\mathcal{E}$, a model $\mathbb{M} : \mathcal{E}$, a security parameter η, a random tape ρ and a memory μ, we write $\mathbb{M}, \eta, \rho, \mu \models^A \varphi$ to denote that φ is satisfied by the left-hand side elements. Note that an assertion φ can specify properties of both the game's memory μ and logical values, including names, thanks to ρ. This allows, e.g., to have an assertion expressing that the value of a particular name $[\![\mathsf{n}\ t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$ does not belong to some list stored in the game's memory μ.

We assume that assertions support propositional connectives, and that local formulas can be seen as assertions. Satisfaction for these constructs should behave as expected, e.g. $\mathbb{M}, \eta, \rho, \mu \models^A f$ iff. $[\![f]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = 1$; and $\mathbb{M}, \eta, \rho, \mu \models^A \varphi \Rightarrow \psi$ iff. $\mathbb{M}, \eta, \rho, \mu \models^A \varphi$ implies $\mathbb{M}, \eta, \rho, \mu \models^A \psi$.

4.3 Bi-deduction Judgement

We now have all the ingredients to form our bi-deduction judgement. Defining its semantics, though, requires a little more work.

Definition 5. *Let $u^1_\#, \ldots, u^m_\#, v_\#$ be a sequence of bi-terms of base types. We say that a program* p *with distinguished variables $X_1, \ldots, X_m$ and* res *computes $\vec{u}_\# \triangleright v_\#$ w.r.t. $\mathbb{M}, \eta, \mathfrak{p}, \rho, \mu$ and side $i \in \{0,1\}$ when:*

$$\mu'[\mathsf{res}] = [\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} \text{ with } \mu' = (\!|\mathsf{p}|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu[X_k \mapsto [\![u^k_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}]_{1 \le k \le m}}$$

$$\mathrm{Fun}(c_1,c_2) \stackrel{\text{def}}{=} \forall\vec{\alpha}_1\forall\vec{\alpha}_2.\ f_1\wedge f_2 \Rightarrow t_1\neq t_2 \quad \text{when } T_1\neq T_2 \text{ and } \mathsf{n}_1=\mathsf{n}_2, \quad \text{and } \mathrm{Fun}(c_1,c_2) \stackrel{\text{def}}{=} \top \text{ otherwise}$$

$$\mathrm{Fresh}(c_1,c_2) \stackrel{\text{def}}{=} \forall\vec{\alpha}_1\forall\vec{\alpha}_2.\ f_1\wedge f_2 \Rightarrow c_1(\vec{\alpha}_1)\neq c_2(\vec{\alpha}_2)\Rightarrow t_1\neq t_2 \quad \text{when } T_1=T_2=\mathsf{T}^{\mathsf{loc}}_{\mathsf{G}},\ \mathsf{n}_1=\mathsf{n}_2, \quad \text{and } \mathrm{Fun}(c_1,c_2) \stackrel{\text{def}}{=} \top \text{ otherwise}$$

$$\mathrm{Unique}(c_1,c_2) \stackrel{\text{def}}{=} \begin{cases} \forall\vec{\alpha}_1\forall\vec{\alpha}_2.\ f_1\wedge f_2\Rightarrow t_1=t_2 & \text{when } T_1=T_2\in\{\mathsf{T}^{\mathsf{glob}}_{\mathsf{G},v}\mid v\in \mathsf{G.gs}\},\ \mathsf{n}_1=\mathsf{n}_2\\ \forall\vec{\alpha}_1\forall\vec{\alpha}_2.\ f_1\wedge f_2\Rightarrow \bot & \text{when } T_1=T_2\in\{\mathsf{T}^{\mathsf{glob}}_{\mathsf{G},v}\mid v\in \mathsf{G.gs}\},\ \mathsf{n}_1\neq\mathsf{n}_2\\ \top & \text{otherwise}\end{cases}$$

where $c_i=(\vec{\alpha}_i,\mathsf{n}_i,t_i,T_i,f_i)$ for i $i\in\{1;2\}$; and we let $c_1(\vec{\alpha}_1)\neq c_2(\vec{\alpha}_2)$ be a shorthand for $\top$ if c_1 and c_2 are distinct occurrences, and $\vec{\alpha}_1\neq\vec{\alpha}_2$ otherwise.

Figure 5: Constraint validity conditions.

In this context, μ' is the final memory *of the computation.*

Naively, one may then say that a bi-deduction $u_\# \triangleright v_\#$ holds w.r.t. a game G when there exists a simulator p against G which computes $u_\# \triangleright v_\#$ w.r.t. any $\mathbb{M}, \eta, \rho, \mathfrak{p}, \mu$ and i. While it makes sense to quantify universally over $\mathbb{M}, \eta, \mu$ and i, doing the same for $\mathfrak{p}$ and ρ would be meaningless, resulting in an unfeasible notion of bi-deduction. Intuitively, we can only expect the semantics of program p and v_i to coincide *if they agree on the parts of the tapes that are read.* Crucially, these parts will be described by the constraint system associated to the considered bi-deduction. For example, if we need a name k to correspond to the PRF game's (globally sampled) key key, it is necessary that the tapes ρ and $\mathfrak{p}$ coincide on positions corresponding to, resp., k (for ρ) and key (for $\mathfrak{p}$).

In order to define this relation between logical and program random tapes, we assume a mapping from (semantic) names to offsets in program random tapes: for each environment $\mathcal{E}$, for each name symbol $\mathsf{n} : \tau' \to \tau$ declared in $\mathcal{E}$, for each $\mathbb{M} : \mathcal{E}$, $\eta \in \mathbb{N}$ and $a \in [\![\tau']\!]^\eta_{\mathbb{M}}$, we assume an offset $O_{\mathbb{M},\eta}(\mathsf{n}, a) \in \mathbb{N}$, such that $(1^\eta, a) \mapsto O_{\mathbb{M},\eta}(\mathsf{n}, a)$ is injective and PTIME computable — this actually corresponds to the $\mathrm{offset}_{\mathsf{n}}(a)$ library function in the simulator of Fig. 2.

Definition 6. *Let C be a constraint system and $\mathbb{M}$ a model, both w.r.t. $\mathcal{E}$. For any $\eta \in \mathbb{N}$, we define $\mathcal{R}^\eta_{C,\mathbb{M}}$ as the relation between $\mathbb{T}_{\mathbb{M},\eta}$ and program random tapes $\mathfrak{P}$ such that $\rho\ \mathcal{R}^\eta_{C,\mathbb{M}}\ \mathfrak{p}$ holds when ρ_a is a prefix of $\mathfrak{p}[\mathsf{T}_{\mathsf{A}}, \mathsf{bool}]$ and for all $(\mathsf{n}, a, \mathsf{T}) \in \mathcal{N}^{\eta,\rho}_{C,\mathbb{M}}$, $[\![\mathsf{n}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(a) = \mathfrak{p}|^\eta_{\mathsf{T}}[O_{\mathbb{M},\eta}(\mathsf{n}, a)]$.*

Couplings between logical and program tapes. Constraining the bi-deduction $\emptyset_\# \triangleright v_\#$ by C will guarantee that there exists a program p which computes $\emptyset \triangleright v_\#$ w.r.t. any tapes $\mathfrak{p}, \rho$ that are related by $\mathcal{R}^\eta_{C,\mathbb{M}}$, i.e. (omitting the initial memory):

$$\text{for all } i\in\{0,1\} \text{ and } \rho\ \mathcal{R}^\eta_{C,\mathbb{M}}\ \mathfrak{p},\ \ (\!|p|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},i}[\mathrm{res}] = [\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}.$$

In order to be able to *lift* the equality above to an equality over *distributions* (required in computational indistinguishability), i.e. to show that for any possible value x,

$$\Pr_{\mathfrak{p}\in\mathfrak{P}}\left((\!|p|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},i}[\mathrm{res}]=x\right) = \Pr_{\rho\in\mathbb{T}_{\mathbb{M},\eta}}\left([\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}=x\right) \tag{3}$$

we rely on the standard notions of probabilistic coupling and lifting (as in [13]). We only present the main intuitions here; see Appendix C.3for details. Consider some distribution $\mathbb{C}$ over *pairs* of logical and program tapes in $\mathbb{T}_{\mathbb{M},\eta} \times \mathfrak{P}$. The *left marginal* of $\mathbb{C}$ is the distribution over $\mathbb{T}_{\mathbb{M},\eta}$ obtained by extracting the logical tape ρ from a pair of tapes $(\rho, \mathfrak{p})$ sampled according to $\mathbb{C}$. The *right marginal* of $\mathbb{C}$ is similar, except that it extracts the program tape $\mathfrak{p}$. A distribution $\mathbb{C}$ is said to be a *probabilistic coupling* of $\mathbb{T}_{\mathbb{M},\eta}$ and $\mathfrak{P}$, which we write $\mathbb{C} : \mathbb{T}_{\mathbb{M},\eta} \bowtie \mathfrak{P}$, if its left and right marginals follow the same distributions as the distributions endowing, resp., $\mathbb{T}_{\mathbb{M},\eta}$ and $\mathfrak{P}$. When $\mathbb{C} : \mathbb{T}_{\mathbb{M},\eta} \bowtie \mathfrak{P}$, we thus have, for any x:

$$\Pr_{\rho\in\mathbb{T}_{\mathbb{M},\eta}}\left([\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}=x\right) = \Pr_{(\rho,\mathfrak{p})\in\mathbb{C}}\left([\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}=x\right) \tag{4}$$

$$\Pr_{\mathfrak{p}\in\mathfrak{P}}\left((\!|p|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},i}[\mathrm{res}]=x\right) = \Pr_{(\rho,\mathfrak{p})\in\mathbb{C}}\left((\!|p|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},i}[\mathrm{res}]=x\right) \tag{5}$$

where the top (resp. bottom) equation follows from the left (resp. right) marginal property of $\mathbb{C}$.

Assume that we can build a coupling $\mathbb{C} : \mathbb{T}_{\mathbb{M},\eta} \bowtie \mathfrak{P}$ contained in $\mathcal{R}^\eta_{C,\mathbb{M}}$ (roughly, this means that $\mathbb{C}$ only samples pairs of tapes related by $\mathcal{R}^\eta_{C,\mathbb{M}}$). Then Eq. (3) holds. Indeed:

$$\begin{aligned}
&\Pr_{\rho\in\mathbb{T}_{\mathbb{M},\eta}}\left([\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}=x\right)\\
&= \Pr_{(\rho,\mathfrak{p})\in\mathbb{C}}\left([\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}=x\right) && \text{(Eq. (4))}\\
&= \Pr_{(\rho,\mathfrak{p})\in\mathbb{C}}\left((\!|p|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},i}[\mathrm{res}]=x\right) && (\mathbb{C}\text{ contained in }\mathcal{R}^\eta_{C,\mathbb{M}})\\
&= \Pr_{\mathfrak{p}\in\mathfrak{P}}\left((\!|p|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},i}[\mathrm{res}]=x\right) && \text{(Eq. (5))}
\end{aligned}$$

Couplings from constraint systems. Given a constraint system C, we would like to build a coupling that is contained in $\mathcal{R}^\eta_{C,\mathbb{M}}$. It turns out that this cannot always be achieved: counter-examples arise when a constraint $c = (\vec{\alpha}, \mathsf{n}, t, \mathsf{T}, f)$ features a condition f (or an index t) that depends on the name n t introduced in the constraint (see Example 9, Appendix C for an explicit counter-example). Such pathological cases are however irrelevant for our use of constraint systems, and we rule them out by introducing in Appendix C.3the notion of *well-formed* constraint system. Given a valid and well-formed constraint system, we are then able to build the desired coupling. Roughly, this is done step by step: well-formedness ensures that there exists an order in which to sample the names corresponding to constraints such that, when processing a constraint c, we are able to compute f and t using the already sampled parts of the tape; then, if f holds, we sample the segments of the logical and program tapes determined by n, t and T (validity ensures that these segments are not yet sampled). Once all constraints are processed, the rest of the tapes is sampled using the relevant probability distributions.

The following key lemma establishes that any well-formed and valid constraint system C can be used to build a coupling contained in $\mathcal{R}^\eta_{C,\mathbb{M}}$ (see proof in Appendix C.5).

Lemma 1. *Let C be a well-formed constraint system w.r.t. $\mathbb{M}, \eta$ such that $\mathbb{M} \models \mathrm{Valid}(C)$. Then, there exists a coupling $\mathbb{C} : \mathbb{T}_{\mathbb{M},\eta} \bowtie \mathfrak{P}$ contained in $\mathcal{R}^{\eta}_{C,\mathbb{M}}$.*

This lemma will be key to justify our Bi-deduce rule, which involves a bi-deduction judgment with empty inputs. However, the notion of well-formedness needs to be adapted to arbitrary bi-deductions: the general notion of well-formedness is relative to the input terms.

Bi-deduction. We finally define our intricate notion of bi-deduction.

Definition 7. *A bi-deduction judgement is of the form:*

$$\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd_{\mathsf{G}} v_\#$$

where G *is a game, $\mathcal{E}$ is an environment, Θ is a set of global formulas, $C_\#$ is a constraint bi-system, the pre-condition $\varphi_\#$ and post-condition $\psi_\#$ are bi-assertions, the inputs $\vec{u}_\#$ is a vector of bi-terms and $v_\#$ is a bi-term.*

It is valid *when, for any type structure $\mathbb{M}_0$, there exists a PTIME program* p *against* G *such that for any model $\mathbb{M} : \mathcal{E}$ extending $\mathbb{M}_0$ in such a way that $\mathbb{M} \models \Theta \tilde{\wedge} \mathrm{Valid}(C_\#)$,* p *is an adversary and for any $\eta \in \mathbb{N}$, $i \in \{0,1\}$, $C_\#$ is well-formed w.r.t. $\mathbb{M}, \eta$ relatively to $\vec{u}_\#$ and for any tapes ρ $\mathcal{R}^{\eta}_{C,\mathbb{M}}$ $\mathfrak{p}$, and for any μ such that $\mathbb{M}, \eta, \rho, \mu \models^{A} \varphi_i$,* p *computes $\vec{u}_\# \rhd v_\#$ w.r.t. $\mathbb{M}, \eta, \mathfrak{p}, \rho, \mu, i$ and the corresponding final memory μ' is such that $\mathbb{M}, \eta, \rho, \mu' \models^{A} \psi_i$. Moreover, we require that the computation of p relies on global samplings $G_\$$ and local samplings $L_\$$ such that*

$$G_\$ \subseteq \{\, O_{\mathbb{M},\eta}(\mathsf{n}, a) \mid \langle \mathsf{n}, a, \mathsf{T}^{\mathrm{glob}}_{\mathsf{G},v} \rangle \in \mathcal{N}^{\eta,\rho}_{c,\mathbb{M}},\ c \in C \,\}$$

$$\text{and}\quad L_\$ \subseteq \{\, O_{\mathbb{M},\eta}(\mathsf{n}, a) \mid \langle \mathsf{n}, a, \mathsf{T}^{\mathrm{loc}}_{\mathsf{G}} \rangle \in \mathcal{N}^{\eta,\rho}_{c,\mathbb{M}},\ c \in C \,\}.$$

Note that, while the general structure of the previous definition is guided by the need to derive indistinguishabilities from bi-deducibilities (as proved formally in the next theorem), some aspects of the definition are not necessary for this goal but ease compositional proofs of bi-deduction through our proof system. This is the case for the conditions on local and global samplings, which make it easy to compose programs while preserving the fact that they are adversaries against G (see Appendix D.3for its proof).

Theorem 1. *Let $\mathcal{E}$ be an environment, Θ a set of global formulas, and $\varphi_\#$ be a bi-assertion such that, for all $\mathbb{M} : \mathcal{E}$ satisfying Θ, for all $i \in \{0,1\}$, η, ρ, we have $\mathbb{M}, \eta, \rho, \mu^{i}_{init\mathbb{M}}{}^{\eta,\rho}(\mathsf{G}) \models^{A} \varphi_i$. The following rule is sound w.r.t. models where* G *is secure, for any $C_\#$, $\#(\vec{v}_0; \vec{v}_1)$ and $\psi_\#$:*

Bi-deduce

$$\frac{\mathcal{E}, \Theta \vdash \mathrm{Valid}(C_\#) \qquad \mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \emptyset \rhd_{\mathsf{G}} \#(\vec{v}_0; \vec{v}_1)}{\mathcal{E}, \Theta \vdash \vec{v}_0 \sim \vec{v}_1}$$

4.4 Proof System

We now present the proof system we designed for bideduction. Our proof rules are guided by the structure of the term to be bideduced. To enable expressive rules, it is useful to consider vectors of *conditional* terms. We will thus consider bi-deductions of the form $\vec{u} \rhd ((f_1, \text{if } f_1 \text{ then } t_1), \ldots, (f_n, \text{if } f_n \text{ then } t_n))$, [1] noted more conveniently $\vec{u} \rhd ((t_1 \mid f_1), \ldots, (t_n \mid f_n))$ or even $\vec{u} \rhd \vec{t}$ when the conditions are irrelevant, using $\vec{t}$ to denote conditioned terms. We present below an overview of the rules of our proof system, providing in Appendix D . the full set of proof rules as well as soundness arguments and an example derivation.

We shall use two operations on constraint systems. First, the concatenation of bi-constraint systems is defined as $\#(C_0^1; C_1^1) \cdot \#(C_0^2; C_1^2) = \#(C_0^1 \cdot C_0^2; C_1^1 \cdot C_1^2)$. Note that the validity of the concatenation of two systems carries over to each of them. Second, we define the generalization $\prod x.C_\#$ of $C_\#$ over x as the system $C_\#$ where x is added to the vector of bound variables in all basic constraints of C_0 and C_1. The validity of $\prod x.C_\#$ implies that of $C_\#$.

We show a selected set of rules in Fig. 6, which we describe below. First, our proof system features rules for basic simulator constructions: Dup, Refl, FA and If-then-else. More interestingly, a central rule of our proof system is Transitivity, which corresponds to composing simulators. To see why it is valid, consider a model $\mathbb{M}$ of Θ and $\mathrm{Valid}(C_\#^1 \cdot C_\#^2)$, and the simulators p_a and p_b provided by the premises. The simulator justifying the bi-deduction in conclusion will be $(\mathsf{p}_a; \mathsf{p}_b)$: additional inputs of the second simulator will be computed by the first one. We can show that this program is also an adversary for the game, and satisfies the conditions on local and global offsets imposed by the bi-deduction semantics. A crucial point here is that, because $C_\#^1 \cdot C_\#^2$ is valid, the freshness conditions for local samplings on the separate executions of p_a and p_b imply the same thing for their sequential composition. Similarly, this validity implies that global samplings are consistent across the two executions. The well-formedness of $C_\#^1 \cdot C_\#^2$ (relatively to $\vec{u}_\#$) does not follow from that of the sub-systems; to prove it, we rely on the specific context of this rule, including the existence of a simulator for the first bi-deduction premise (see Appendix D.2.1 for details)..

In order to represent unbounded collections of objects to bideduce, we extend the bi-deduction judgement beyond terms of base type, allowing order-1 types when the argument types are enumerable — this is captured by the type restriction enum, cf. Appendix D.1.. This does not change the semantics of bi-deduction: we simply view these functions as an explicit representation of their graph. This extension notably brings in the Lambda and Induction rules of Fig. 6, the latter allowing proofs of bi-deduction by induction. In both cases, we require the pre- and post-conditions to coincide: this is because the underlying simulator computation iterates the computation of the simulator corresponding to the premise; the condition on the game's memory must be invariant through this iteration. In the induction rule, the type restriction well-founded$_\tau(<)$ states that $([\![\tau]\!]^{\eta}_{\mathbb{M}}, [\![<]\!]^{\eta}_{\mathbb{M},\mathcal{E}})$ is well-founded.

Two rules introduce new constraints in their conclusion. The first rule, Name, allows a simulator to sample a name. The second rule is for oracle calls, and requires a preliminary definition before we introduce it in Proposition 1. An *oracle triple* for an oracle f, written $\{\varphi_\#\} v_\# \leftarrow O_f(\vec{t}_\#)[\vec{k}_\#; \vec{r}_\#]\{\psi_\#\}$ is formed from: assertions $\varphi_\#$ and $\psi_\#$ for the pre- and post-conditions, an output term $v_\#$, input terms $\vec{t}_\#$, and terms $\vec{k}_\#$ and $\vec{s}_\#$ for the global and local randomness offsets of the oracle. We require that the offsets are of the form $\vec{k}_\# = (\mathsf{k}_v\ o_{v\sharp})_{v \in f.\mathrm{glob}_\$}$ and $\vec{r}_\# = (\mathsf{r}_v\ s_{v\sharp})_{v \in f.\mathrm{loc}_\$}$, where k_v and r_v are names. Such a triple is valid, when the oracle called with the specified parameters in a memory satisfying $\varphi_\#$, returns $v_\#$ in a memory satisfying $\psi_\#$ (details in Appendix D.2.4)..

[1] Here, (if f then t) is syntactic sugar for (if f then t else witness$_\tau$) where τ is the type of t and witness$_\tau$ is an arbitrary symbol of type τ (whose existence is guaranteed, as all types are inhabited).

$$\textsc{Dup}\quad \frac{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\#\rhd \vec{v}_\#,\vec{t}_\#}{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\#\rhd \vec{v}_\#,\vec{t}_\#,\vec{t}_\#}$$

$$\textsc{Transitivity}\quad \frac{\mathcal{E},\Theta,C^1_\#,(\varphi_\#,\varphi'_\#)\vdash \vec{u}_\#\rhd \vec{t}_\#\qquad \mathcal{E},\Theta,C^2_\#,(\varphi'_\#,\psi_\#)\vdash \vec{u}_\#,\vec{t}_\#\rhd \vec{v}_\#}{\mathcal{E},\Theta,C^1_\#\cdot C^2_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\#\rhd \vec{t}_\#,\vec{v}_\#}$$

$$\textsc{Refl}\quad \frac{}{\mathcal{E},\Theta,\emptyset,(\varphi_\#,\varphi_\#)\vdash \vec{u}_\#,t_\#\rhd t_\#}$$

$$\textsc{FA}\quad \frac{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\#\rhd \vec{v}_\#,(t^1_\#\mid f_\#),\ldots,(t^n_\#\mid f_\#)\qquad \mathcal{E},\Theta\vdash \mathsf{adv}(g)}{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\#\rhd \vec{v}_\#,(g\ t^1_\#\ \ldots\ t^n_\#\mid f_\#)}$$

$$\textsc{If-then-else}\quad \frac{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\#\rhd \vec{v}_\#,(b_\#\mid f_\#),(t_\#\mid f_\#\wedge b_\#),(t'_\#\mid f_\#\wedge\neg b_\#)}{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\#\rhd \vec{v}_\#,(\text{if } b_\# \text{ then } t_\# \text{ else } t'_\#\mid f_\#)}$$

$$\textsc{Lambda}\quad \frac{(\mathcal{E},x:\tau),\Theta,C_\#,(\varphi_\#,\varphi_\#)\vdash \vec{u}_\#,x\rhd (t_\#\mid f_\#)\qquad \mathcal{E},x:\tau\vdash t_\#:\tau_\mathsf{b}\quad \tau_\mathsf{b}\in\mathbb{B}\quad \mathsf{enum}(\tau)}{\mathcal{E},\Theta,\prod_{(x:\tau)}.C_\#,(\varphi_\#,\varphi_\#)\vdash \vec{u}_\#\rhd (\lambda(x:\tau).t_\#\mid f_\#)}$$

$$\textsc{Induction}\quad \frac{(\mathcal{E},x:\tau),\Theta,C_\#,(\varphi_\#,\varphi_\#)\vdash \vec{u}_\#,(\lambda(y:\tau).\text{if } y<x \text{ then } t[x\mapsto y]\mid f_\#),x\rhd (t_\#\mid f_\#)\qquad \mathcal{E},x:\tau\vdash t_\#:\tau_\mathsf{b}\quad \tau_\mathsf{b}\in\mathbb{B}\quad \mathsf{finite}(\tau)\quad \mathsf{fixed}(\tau)\quad \mathcal{E},\Theta\vdash \text{well-founded}_\tau(<)\ \tilde{\wedge}\ \mathsf{adv}(<)}{\mathcal{E},\Theta,\prod_{(x:\tau)}C_\#,(\varphi_\#,\varphi_\#)\vdash \vec{u}_\#\rhd (\lambda(x:\tau).t_\#\mid f_\#)}$$

$$\textsc{Name}\quad \frac{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\#\rhd (t_\#\mid f_\#)}{\mathcal{E},\Theta,C_\#\cdot\{(\emptyset,n,t_\#,\mathsf{T}_S,f_\#)\},(\varphi_\#,\psi_\#)\vdash \vec{u}_\#\rhd (n\ t_\#\mid f_\#)}$$

Figure 6: Selected set of rules.

Proposition 1. *Let* G *be a game and* $f\in O$ *one of its oracles. The following rule is sound w.r.t. the class of models satisfying* G*, using the notations introduced above:*

$$\textsc{Oracle}_f\quad \frac{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\#\rhd_\mathsf{G}\vec{w}_\#,(\vec{t}_\#\mid F_\#),(\vec{o}_\#\mid F_\#),(\vec{s}_\#\mid F_\#)\qquad \Theta\models\{\psi_\#\wedge F_\#\}v_\#\leftarrow O_f(\vec{t}_\#)[\vec{k}_\#;\vec{r}_\#]\{\theta_\#\}}{\mathcal{E},\Theta,C'_\#,(\varphi_\#,\theta_\#)\vdash \vec{u}_\#\rhd_\mathsf{G}\vec{w}_\#,(v_\#\mid F_\#)}$$

with $C'_\# = C_\#\cdot\prod_{v\in f.\mathsf{glob_s}}(\emptyset,\mathsf{k}_v,o_{v\sharp},\mathsf{T}^{\mathsf{glob}}_{\mathsf{G},v},F_\#)\cdot\prod_{v\in f.\mathsf{loc_s}}(\emptyset,\mathsf{r}_v,s_{v\sharp},\mathsf{T}^{\mathsf{loc}}_{\mathsf{G}},F_\#)$

$\vec{o}_\# = (o_{v\sharp})_{v\in f.\mathsf{glob_s}}$ *and* $\vec{s}_\# = (s_{v\sharp})_{v\in f.\mathsf{glob_s}}$

4.5 Examples

We illustrate how our proof system operates using two examples.

Example 5. *Using the PRF game of Fig. 1, we should have that:*

$$(\emptyset,\emptyset)\rhd h(n,\mathsf{k}),\#(\mathsf{h}(m,\mathsf{k});\mathsf{n}_\mathsf{fresh})$$

for any adversarial messages n,m *such that* n *and* m *are always distinct, i.e.* $[n\neq m]_\mathsf{e}$.

If n *and* m *are two names* n *and* m*, we cannot guarantee that they are* always *distinct. However, we have the following:*

$$(\emptyset,\emptyset)\rhd \mathsf{h}(\mathsf{n},\mathsf{k}),\text{if } \mathsf{n}\neq\mathsf{m} \text{ then } \#(\mathsf{h}(\mathsf{m},\mathsf{k});\mathsf{n}_\mathsf{fresh})$$

Here, the challenge oracle is only called in the then branch, when $\mathsf{n}\neq\mathsf{m}$ *does hold. The ability to propagate information from term-level conditionals to assertions is crucial to verify such bi-deductions.*

Example 6. *We are going to show the indistinguishability*

$$\mathsf{h}(\mathsf{n},\mathsf{k}),\mathsf{h}(\mathsf{m},\mathsf{s}),\mathsf{h}(\mathsf{m},\mathsf{k})\sim\mathsf{h}(\mathsf{n},\mathsf{k}),\mathsf{h}(\mathsf{m},\mathsf{s}),\mathsf{n}_\mathsf{fresh}\qquad(6)$$

using bi-deduction w.r.t. the PRF game of Fig. 1, where n, k, m, s *are distinct names of type* τ*. We are going to show that:*

(1) *the first and last terms can be computed using oracle calls;*
(2) *and that the middle term is just a function application.*

For Item 2, we require that h *is an adversarial function symbol. To be able to carry out the simulation strategy of Item 1, we need to ensure that* $\mathsf{n}\neq\mathsf{m}$*, which we do using the trick of Example 5. First, we assume that the type* τ *is a* large type*: essentially, this means that independent names with output type* τ *have a negligible probability of collision. This assumption is captured by the* $\mathsf{large}(\tau)$ *hypothesis introduced in [5]. Under this hypothesis, it can be shown that* $\mathsf{large}(\tau)\models\mathsf{n}\neq\mathsf{m}\sim$ *true and thus, using the rewriting rule of [5], that the indistinguishability in Eq.* (6) *is implied by the formula:*

$$\mathsf{h}(\mathsf{n},\mathsf{k}),\mathsf{h}(\mathsf{m},\mathsf{s}),\text{if } \mathsf{n}\neq\mathsf{m} \text{ then } \mathsf{h}(\mathsf{m},\mathsf{k})\sim\mathsf{h}(\mathsf{n},\mathsf{k}),\mathsf{h}(\mathsf{m},\mathsf{s}),\text{if } \mathsf{n}\neq\mathsf{m} \text{ then } \mathsf{n}_\mathsf{fresh}$$

Let's take the hypotheses $\Theta=\mathsf{adv}(\mathsf{h}),\mathsf{large}(\tau)$.

For the sake of simplicity, we instantiate assertions by sets of memories: thus, roughly, satisfiability is set membership and implication is set inclusion. Consider the following assertions:

$$\varphi_0 = \{(l_\mathsf{hash}\mapsto[]\ ;\ l_\mathsf{challenge}\mapsto[])\}$$
$$\varphi = \{(l_\mathsf{hash}\mapsto[\mathsf{n}]\ ;\ l_\mathsf{challenge}\mapsto[])\}$$

where [] *is the empty list and* [n] *is the list containing a single element* n*. We have:*

$$\Theta\models\{\varphi_0\}\,\mathsf{h}(\mathsf{n},\mathsf{k})\leftarrow O_\mathsf{hash}(\mathsf{n})[\mathsf{k};.]\,\{\varphi\},$$

and, using Name *to compute* n*, we get the bi-deduction judgment:*

$$\Theta,\big((\mathsf{n},\mathsf{T}_S,\top),(\mathsf{k},\mathsf{T}^{\mathsf{glob}}_{\mathsf{G},\mathsf{k}},\top)\big),(\varphi_0,\varphi)\vdash\emptyset\rhd\mathsf{h}(\mathsf{n},\mathsf{k}).$$

All the name constraints in this example have no bound variables and are for names n, k, s, m *which are not indexed. Thus, we use a shorter syntax for constraints, and write* $(\mathsf{name},\mathsf{T},f)$ *instead of* $(\emptyset,\mathsf{name},\emptyset,\mathsf{T},f)$.

Then, using Name*,* Dup *and* FA*, we also get that:*

$$\Theta,\big((\mathsf{m},\mathsf{T}_S,\top),(\mathsf{s},\mathsf{T}_S,\top)\big),(\varphi,\varphi)\vdash\emptyset\rhd\mathsf{h}(\mathsf{m},\mathsf{s}).$$

Finally, we have:

$$\{\varphi\wedge\mathsf{n}\neq\mathsf{m}\}\ \#(\mathsf{h}(\mathsf{m},\mathsf{k});\mathsf{n}_\mathsf{fresh})\leftarrow O_\mathsf{challenge}(\mathsf{m})[\mathsf{k};\mathsf{n}_\mathsf{fresh}]\ \{\psi\}$$

for a certain ψ*. As before, using the rules* If-then-else*,* $\textsc{Oracle}_f$ *for* $f=\mathsf{challenge}$ *and* Name*, we have that:*

$$\Theta,\big((\mathsf{n},\mathsf{T}_S,\top),(\mathsf{m},\mathsf{T}_S,\top),(\mathsf{m},\mathsf{T}_S,\mathsf{n}\neq\mathsf{m})$$
$$(\mathsf{k},\mathsf{T}^{\mathsf{glob}}_{\mathsf{G},\mathsf{k}},\mathsf{n}\neq\mathsf{m}),(\mathsf{n}_\mathsf{fresh},\mathsf{T}^{\mathsf{loc}}_{\mathsf{G}},\mathsf{n}\neq\mathsf{m})\big),(\varphi,\psi)\vdash$$
$$\emptyset\rhd\text{if } \mathsf{n}\neq\mathsf{m} \text{ then } \#(\mathsf{h}(\mathsf{m},\mathsf{k});\mathsf{n}_\mathsf{fresh}).$$

By transitivity, we get the final judgement:

$$\mathcal{E}, \Theta, C, (\varphi_0, \psi) \vdash \emptyset \rhd \mathsf{h}(\mathsf{n}, \mathsf{k}), \mathsf{h}(\mathsf{m}, \mathsf{s}), \text{if } \mathsf{n} \neq \mathsf{m} \text{ then } \#(\mathsf{h}(\mathsf{m}, \mathsf{k}); \mathsf{n}_{\mathsf{fresh}})$$

where C is the following constraint system:

$$\{\ (\mathsf{n}, \mathsf{T}_S, \top) \quad , (\mathsf{k}, \mathsf{T}^{\mathsf{glob}}_{\mathsf{G},\mathsf{k}}, \top) \quad , (\mathsf{m}, \mathsf{T}_S, \top),$$
$$(\mathsf{s}, \mathsf{T}_S, \top) \quad , (\mathsf{n}, \mathsf{T}_S, \top) \quad , (\mathsf{m}, \mathsf{T}_S, \top),$$
$$(\mathsf{m}, \mathsf{T}_S, \mathsf{n} \neq \mathsf{m}),\ (\mathsf{k}, \mathsf{T}^{\mathsf{glob}}_{\mathsf{G},\mathsf{k}}, \mathsf{n} \neq \mathsf{m}),\ (\mathsf{n}_{\mathsf{fresh}}, \mathsf{T}^{\mathsf{loc}}_{\mathsf{G}}, \mathsf{n} \neq \mathsf{m})\ \}$$

Then $\mathrm{Valid}(C)$ *is easily verified, and the proof is done.*

5 PROOF SEARCH AND IMPLEMENTATION

We now describe the specification, heuristic and design choices of our proof-search procedure — called search$_\rhd$ — for bi-deduction, as well as its implementation in our extension of Squirrel [27]. Our extension allows users to specify arbitrary games, and bi-deduction verification is made available to the users through a tactic **crypto** based on search$_\rhd$, which takes as input the game to be used and some (optional) initial constraints. Upon success, the tactic reduces the equivalence to be proved to proof obligations corresponding to missing parts of the constructed bi-deduction derivation.

Scope. Our goal is for search$_\rhd$ and **crypto** to reach the expressivity level of Squirrel legacy cryptographic tactics, while being able to tackle new cryptographic games. Crucially, legacy cryptographic tactics, as well as **crypto**, are not expected to apply in complex scenarios: a typical Squirrel proofs consists in modifying the proof-goal using its indistinguishability logic [8] to pave the way for the application of a cryptographic game. Furthermore, since Squirrel is an interactive proof assistant, we aim for **crypto** to have a low running time (i.e. a few seconds). This led to the following design choice: search$_\rhd$ does not backtrack and does not handle induction. The former limitation is partially alleviated by a careful design of our heuristics, and the latter with ad hoc pre-processing in the implementation (described later).

Proof-search. Our procedure take as input a partial bi-deduction judgment which it tries to complete into a valid judgement. That is, given an environment $\mathcal{E}$, hypotheses Θ, inputs $\vec{u}_\#$, a bi-constraint system $C_\#$ well-formed relatively to $\vec{u}_\#$, a pre-condition $\varphi_\#$, and outputs $\vec{v}_\#$, search$_\rhd$ looks for additional hypotheses Θ', constraints $C'_\#$, and a post-condition $\psi_\#$, such that the following bi-deduction judgement is valid:

$$\mathcal{E}, \Theta \cup \Theta', C_\# \cdot C'_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd \vec{v}_\#$$

Heuristic. The procedure proceeds by backwards proof search, applying rules whose conclusion matches $\vec{v}_\#$ and recursing on the rule premises. It greedily applies the oracle rule, but avoids using it in scenarios where name constraints added by the oracle rule would trivially lead to an unsatisfiable constraint system. E.g., search$_\rhd$ will not apply the oracle rule if doing so would associate a name with a global tag $\mathsf{T}^{\mathsf{glob}}_{\mathsf{G},v}$ when this tag is already associated to a different name. When the status of a name w.r.t. constraints does not necessarily allow to use an oracle, search$_\rhd$ rewrites the term to deduce in order to conditionally apply the oracle. For example, in the PRF game using name $\mathsf{k}\ i$ as the key, we can only obtain $\mathsf{h}(m, \mathsf{k}\ j)$ using the hash oracle when $i = j$ (assuming that m is not in the logs). In that case, search$_\rhd$ rewrites the term into if $i = j$ then $\mathsf{h}(m, \mathsf{k}\ i)$ else $\mathsf{h}(m, \mathsf{k}\ j)$, and uses the bi-deduction rule for conditionals, to conclude using the hash oracle in the case where $i = j$, and by computing the message explicitly otherwise (adding the constraint that $\mathsf{k}\ j$ is a simulator name when $j \neq i$).

Implementation. In addition to an implementation of search$_\rhd$, the **crypto** tactic comes with a new syntax to declare arbitrary games, an instantiation of the assertion logic, and a pre-processing technique to handle recursive terms. Our implementation of the assertion logic only supports sets of messages, which allows to handle, e.g., the sets of hashed messages in the PRF game of Fig. 1. As we shall see, this suffices to support the games corresponding to legacy cryptographic axioms, as well as some new (standard) games. Extending the assertion logic beyond that is left to future work.

We designed a pre-processing technique to handle recursive terms. When **crypto** is called on a equivalence $\vec{v_0} \sim \vec{v_1}$, it first tries to show that all recursive sub-terms of $\vec{v_0} \sim \vec{v_1}$ can be bi-deduced by induction. To do so, it generates a partial bi-deduction sub-goal corresponding to the premise of the induction rule, and calls search$_\rhd$ on this partial sub-goal until it completes into a bi-deduction judgement with a fixed-point assertion on the game's memory (i.e. $\varphi_\# = \psi_\#$). Then, **crypto** calls search$_\rhd$ on the initial bi-deduction sub-goal $\#(\vec{v_0}; \vec{v_1})$, knowing (by transitivity) that all recursive sub-terms of $\#(\vec{v_0}; \vec{v_1})$ can be bi-deduced. At the end of its execution, a standard proof-obligation is generated to guarantee that the constraint system returned by search$_\rhd$ is valid. See Appendix E.1 for more details on how recursion is handled.

6 CASE STUDIES

Our implementation has allowed us to validate our approach on several promising case studies, which we briefly describe below. These case-studies are available with the source code of the tool (in sub-directory `case-studies-ccs/`), as well as in HTML files that allow to replay the runs of Squirrel on each example without installing the tool.

The file `hash-lock.sp` presents the Squirrel proof of our running example, i.e. strong secrecy for the Hash Lock protocols, derived from the PRF game as in our examples.

We then illustrate how our **crypto** tactic can eventually replace existing tactics, on the example of the Basic Hash protocol, which is proved unlinkable in existing case studies using the EUF and PRF legacy tactics. We adapt the same arguments using **crypto** with both EUF and PRF games in file `basic-hash.sp`. This shows that our bi-deduction verification is already powerful enough for real examples, though there is some work to be done to make it as easy to use as legacy crypto tactics.

We show, obviously, that our approach is not limited to cryptographic assumptions already supported by Squirrel. In the file `private-authentication.sp` we prove anonymity of the Private Authentication protocol [20] using a previously unsupported cryptographic assumption, $\mathrm{CPA}_\$$, which roughly states the indistinguishability between an encrypted message and a fresh random sampling. In the file `nsl.sp`, we prove a key result about the Needham-Schroeder-Lowe public-key protocol [38], which crucially relies on the CCA_2 game that was previously unsupported in Squirrel.

Legacy cryptographic tactics in Squirrel can only handle #(_; _) in indistinguishabilities, and not in the protocols. Interestingly, **crypto** does not have this limitation: in `global-cpa.sp`, we prove the equivalence between two protocols outputting different values (of the same length) using **crypto** on the CPA game; such equivalences are often useful when reasoning about protocols.

7 RELATED WORK AND DISCUSSION

We compare our work to different approaches in the area of programming languages and formal methods for cryptography in Section 7.1, 7.2 and 7.3, and discuss trust assumptions in Section 7.4.

7.1 Mechanizing cryptographic reductions

Different techniques have been used to obtain formal mechanized proofs of cryptographic arguments.

Program logics. Techniques [2, 15, 16] relying on imperative program logics, the most prominent one being the probabilistic relational Hoare Logic, encode the cryptographic design and security property under study as a stateful and sequential imperative program. Then, the cryptographic arguments proving this program security can be captured by program logics. These approaches are very expressive, but current tools only support the manual application of cryptographic games: to reduce the security of a design Π to a game G, one has to explicitly write a simulator $\mathcal{S}$ such that $\Pi = \mathcal{S}^{\mathrm{G}}$. We do not have this limitation.

Often, these approaches embed their program logic in an expressive ambient logic, e.g. SSProve [2] is a Coq framework, and EasyCrypt [15] implements a higher-order ambient logic. While Squirrel's local logic is also a higher-order logic, its (current) global logic is less expressive than, e.g., Easycrypt's ambient logic, because it relies on asymptotic rather than concrete security — though recent work [7] blurs this demarcation. This is deliberate: Squirrel aims to capture higher-level arguments, with a focus on protocols, which are notoriously laborious to analyze in pRHL-based tools. Because of this, past Squirrel developments have required less mathematical libraries than proofs dealing with crypto primitives.

Game transformations. CryptoVerif [18] is the only tool that automatically finds cryptographic reductions without being restricted to a fixed set of built-in assumptions. CryptoVerif directly manipulates cryptographic games which are iteratively modified using an ad hoc set of game transformations implemented in the tool. Because of its lack of logical foundations, CryptoVerif does not support generic mathematical reasoning. In particular, proof obligations resulting from the application of a cryptographic assumption cannot be discharged to the user as we do, limiting the tool's expressiveness. Moreover, CryptoVerif can only handle assumptions of the form $(\mathrm{G}_0, \mathrm{G}_1)$ where G_0 is a *stateless* game and G_1 features monotonous state (in the form of global write-once tables). Our approach does not suffer from such a restriction from a theoretical point-of-view: arbitrary stateful operations can be handled by using a suitable assertion logic.

Property-specific approaches. There has been some number of works which aim at automating cryptographic proofs for a fixed target security property and a restricted class of programs, e.g. to show that padding-based encryption schemes are IND-CCA$_2$ [12], to prove that block-cipher modes are IND-CPA [39] or AEAD [31], or to analyze the EUF-MAC security of structure preserving signatures [14]. Another similar previous work is Owl [29], which uses a type-based approach to prove reachability properties under a fixed set of cryptographic assumptions (IND-CPA, RO, ...). The restrictions on the class of programs, assumptions and target security properties allow these approaches to be highly automated and efficient, but make them unsuitable as general-purpose frameworks to mechanize cryptographic reductions.

Other CCSA-based approach. CryptoVampire [32] is a recent tool designed to prove trace properties of security protocols using the CCSA framework. CryptoVampire aims for a higher level of automation than Squirrel, by encoding the security of a protocol as a first-order logic task that is then discharged to first-order theorem provers (e.g. Vampire [37]). Because CryptoVampire relies on the standard CCSA crypto axioms, it suffers from the issue we address in this paper.

7.2 Deduction problem

The deduction problem has been extensively studied in the literature, albeit in different settings. E.g. [21, 24, 42] study this problem in Dolev-Yao models, hence they only consider adversaries with very restricted computing capabilities and which do not have access to any oracles. In [23], the authors rely on a deduction predicate with a computational semantics, which they use to prove some security properties. However, this work is mostly interested in non-deducibility rather than deducibility, and they only consider adversaries without access to any oracles.

7.3 Component-based synthesis

Component-based synthesis consists in automatically generating code implementing a given target API, starting from a source API. While our problem could be reformulated in this setting (the target API is the protocol under study, the source API is the cryptographic game), existing CBS techniques are (to the best of our knowledge) unsuitable for our setting, either because the code they can synthesize is too simple for our simulators (e.g. [30, 33] only support loop-free programs) or because they are test-driven and do not provide formal guarantees on the produced code (e.g. [28, 41]). More generally, while the problem of program synthesis has been extensively explored by the programming language community, it is usually done with different goals in mind, and under different design constraints. We are not aware of any work allowing to synthesize recursive and probabilistic programs interacting with stateful APIs in an automated fashion, which is what we need here.

7.4 Trust Assumptions

In order to trust a Squirrel proof, one has to trust several components. At a theoretical level, one first has to trust proof rules. In particular, on has to trust the rules that encode cryptographic assumptions, which is problematic due to their complexity. Our work improves on this by replacing the trust placed in cryptographic rules by trust in the bi-deduction proof rules, and in the cryptographic games on which bi-deduction is instantiated. The latter might still be considered problematic, in particular when non-standard variants of cryptographic games are used to ease

bi-deduction. An interesting perspective in that respect would be to verify formally that the security of the game variants does follow from that of standard games. This can be done by translating the non-standard game to a formal framework where it may be proved from the standard game: that framework could be Squirrel itself (the game would be translated as a protocol) or another tool such as CryptoVerif or EasyCrypt. The latter option would be similar to the recent work on CV2EC [19], which allows to verify CryptoVerif game transformations in EasyCrypt.

Trust is not only placed in theoretical systems, but also in (parts of) the software that implements these systems: the trusted code base (TCB). In the case of Squirrel, the TCB is essentially the whole code base. The same holds for CryptoVerif or EasyCrypt, but foundational frameworks such as SSProve [2] and CryptHOL [16] clearly have a much smaller TCB. In order to improve on this, it would be desirable that Squirrel tactics actually build proof objects from elementary rules; these proof objects could then be verified by a small kernel, as is the case in e.g. Coq, or even by an independent software. In the case of our crypto tactic, the current implementation determines whether a proof exists, but does not build it; if proof search produced a proof object, we would not have to trust the correctness of the proof search algorithm, but could verify its output a posteriori.

To go even further, the proof checker itself could be verified, and integrated in a larger proof development carried out in a foundational proof assistant such as Coq or Isabelle, where Squirrel's elementary proof rules could also be proved sound. Reaching this level of trust would require significant efforts, which do not seem justified at this point. This should be re-evaluated when Squirrel reaches a mature and more stable state, and allows proofs of industrial-scale protocols such as TLS or Signal.

8 CONCLUSION

In order to systematically derive indistinguishabilities from cryptographic games in the tool Squirrel, we have designed a strong notion of bi-deduction, a bi-deduction proof system, and we have implemented an automated proof search procedure for it. We validated this procedure on several case studies.

This promising development calls for several future works. Much work is obviously left to encapsulate bi-deduction verification into user-friendly crypto tactics, and to improve the performance and precision of our verification procedure. We will also push our implementation on larger case studies, including ones using complex, currently unsupported cryptographic assumptions, e.g. from electronic voting. From a theoretical point of view, our bi-deduction relies on an exact semantics (i.e. the simulator must compute the target terms with probability 1) which complicates bi-deduction proofs because it does not allow the (otherwise ubiquitous) reasoning up-to overwhelming equality; we will explore several ways to relax this limitation.

ACKNOWLEDGMENTS

This work received funding from the France 2030 program managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0006.

REFERENCES

[1] 2013. CVE-2014-0160 aka. the Heartbleed bug. Available from MITRE. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160
[2] Carmine Abate, Philipp G. Haselwarter, Exequiel Rivas, Antoine Van Muylder, Théo Winterhalter, Catalin Hritcu, Kenji Maillard, and Bas Spitters. 2021. SSProve: A Foundational Framework for Modular Cryptographic Proofs in Coq. In *CSF*. IEEE, 1–15.
[3] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. 2015. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. In *22nd ACM Conference on Computer and Communications Security*.
[4] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuéllar, Giancarlo Pellegrino, and Alessandro Sorniotti. 2013. An authentication flaw in browser-based single sign-on protocols: Impact and remediations. *Computers & Security* 33 (2013), 41–58.
[5] David Baelde, Stéphanie Delaune, Charlie Jacomme, Adrien Koutsos, and Solène Moreau. 2021. An Interactive Prover for Protocol Verification in the Computational Model. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 537–554.
[6] David Baelde, Stéphanie Delaune, Adrien Koutsos, and Solène Moreau. 2022. Cracking the Stateful Nut: Computational Proofs of Stateful Security Protocols using the Squirrel Proof Assistant. In *CSF*. IEEE, 289–304.
[7] David Baelde, Caroline Fontaine, Adrien Koutsos, Guillaume Scerri, and Théo Vignon. 2024. A Probabilistic Logic for Concrete Security. In *2024 IEEE 37th Computer Security Foundations Symposium (CSF)*. IEEE Computer Society, Los Alamitos, CA, USA, 484–499.
[8] David Baelde, Adrien Koutsos, and Joseph Lallemand. 2023. A Higher-Order Indistinguishability Logic for Cryptographic Reasoning. In *LICS'23*. ACM. https://inria.hal.science/hal-03981949 to appear.
[9] Gergei Bana, Rohit Chadha, and Ajay Kumar Eeralla. 2018. Formal Analysis of Vote Privacy Using Computationally Complete Symbolic Attacker. In *ESORICS (2) (Lecture Notes in Computer Science, Vol. 11099)*. Springer, 350–372.
[10] Gergei Bana and Hubert Comon-Lundh. 2014. A Computationally Complete Symbolic Attacker for Equivalence Properties. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM, 609–620.
[11] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. 2021. SoK: Computer-Aided Cryptography. In *2021 IEEE Symposium on Security and Privacy (SP)*. 777–795.
[12] Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, Yassine Lakhnech, Benedikt Schmidt, and Santiago Zanella Béguelin. 2013. Fully automated analysis of padding-based encryption in the computational model. In *CCS*. ACM, 1247–1260.
[13] Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, Léo Stefanesco, and Pierre-Yves Strub. 2015. Relational Reasoning via Probabilistic Coupling. In *LPAR (Lecture Notes in Computer Science, Vol. 9450)*. Springer, 387–401.
[14] Gilles Barthe, Edvard Fagerholm, Dario Fiore, Andre Scedrov, Benedikt Schmidt, and Mehdi Tibouchi. 2016. Strongly-optimal structure preserving signatures from Type II pairings: synthesis and lower bounds. *IET Inf. Secur.* 10, 6 (2016), 358–371.
[15] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. 2011. Computer-Aided Security Proofs for the Working Cryptographer. In *CRYPTO (Lecture Notes in Computer Science, Vol. 6841)*. Springer, 71–90.
[16] David A. Basin, Andreas Lochbihler, and S. Reza Sefidgar. 2020. CryptHOL: Game-Based Proofs in Higher-Order Logic. *J. Cryptol.* 33, 2 (2020), 494–566.
[17] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. 2017. Verified models and reference implementations for the TLS 1.3 standard candidate. In *2017 IEEE Symposium on Security and Privacy*. IEEE, 483–502.
[18] Bruno Blanchet. 2008. A Computationally Sound Mechanized Prover for Security Protocols. *IEEE Trans. Dependable Secur. Comput.* 5, 4 (2008), 193–207.
[19] Bruno Blanchet, Pierre Boutry, Christian Doczkal, Benjamin Grégoire, and Pierre-Yves Strub. 2024. CV2EC: Getting the Best of Both Worlds. In *2024 IEEE 37th Computer Security Foundations Symposium (CSF)*. IEEE Computer Society, Los Alamitos, CA, USA, 283–298.
[20] Michael Burrows, Martin Abadi, and Roger Needham. 1990. A Logic of Authentication. *ACM Trans. Comput. Syst.* 8, 1 (feb 1990), 18–36.
[21] Sergiu Bursuc, Hubert Comon-Lundh, and Stéphanie Delaune. 2014. Deducibility constraints and blind signatures. *Inf. Comput.* 238 (2014), 106–127.
[22] Hubert Comon and Adrien Koutsos. 2017. Formal Computational Unlinkability Proofs of RFID Protocols. In *CSF*. IEEE Computer Society, 100–114.
[23] Hubert Comon-Lundh, Véronique Cortier, and Guillaume Scerri. 2013. Tractable Inference Systems: An Extension with a Deducibility Predicate. In *CADE (Lecture Notes in Computer Science, Vol. 7898)*. Springer, 91–108.

[24] Hubert Comon-Lundh and Vitaly Shmatikov. 2003. Intruder Deductions, Constraint Solving and Insecurity Decision in Presence of Exclusive or. In *LICS*. IEEE Computer Society, 271.

[25] Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL*. ACM, 238–252.

[26] Cas Cremers, Caroline Fontaine, and Charlie Jacomme. 2022. A Logic and an Interactive Prover for the Computational Post-Quantum Security of Protocols. In *SP*. IEEE, 125–141.

[27] The Squirrel development team. 2024. The Squirrel Prover repository. https://github.com/squirrel-prover/squirrel-prover/.

[28] Yu Feng, Ruben Martins, Yuepeng Wang, Isil Dillig, and Thomas W. Reps. 2017. Component-based synthesis for complex APIs. In *POPL*. ACM, 599–612.

[29] J. Gancher, S. Gibson, P. Singh, S. Dharanikota, and B. Parno. 2023. OWL: Compositional Verification of Security Protocols via an Information-Flow Type System. In *2023 2023 IEEE Symposium on Security and Privacy (SP) (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 1130–1147.

[30] Sumit Gulwani, Susmit Jha, Ashish Tiwari, and Ramarathnam Venkatesan. 2011. Synthesis of loop-free programs. In *PLDI*. ACM, 62–73.

[31] Viet Tung Hoang, Jonathan Katz, and Alex J. Malozemoff. 2015. Automated Analysis and Synthesis of Authenticated Encryption Schemes. In *CCS*. ACM, 84–95.

[32] Simon Jeanteur, Laura Kovács, Matteo Maffei, and Michael Rawson. 2024. CryptoVampire: Automated Reasoning for the Complete Symbolic Attacker Cryptographic Model. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 259–259.

[33] Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. 2010. Oracle-guided component-based program synthesis. In *ICSE (1)*. ACM, 215–224.

[34] Matthieu Journault, Antoine Miné, and Abdelraouf Ouadjaout. 2019. An Abstract Domain for Trees with Numeric Relations. In *ESOP (Lecture Notes in Computer Science, Vol. 11423)*. Springer, 724–751.

[35] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. 2019. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1–19.

[36] Adrien Koutsos. 2019. The 5G-AKA Authentication Protocol Privacy. In *EuroS&P*. IEEE, 464–479.

[37] Laura Kovács and Andrei Voronkov. 2013. First-Order Theorem Proving and Vampire. In *CAV (Lecture Notes in Computer Science, Vol. 8044)*. Springer, 1–35.

[38] Gavin Lowe. 1996. Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 147–166.

[39] Alex J. Malozemoff, Jonathan Katz, and Matthew D. Green. 2014. Automated Analysis and Synthesis of Block-Cipher Modes of Operation. In *CSF*. IEEE Computer Society, 140–152.

[40] David Monniaux. 2003. Abstracting cryptographic protocols with tree automata. *Sci. Comput. Program.* 47, 2-3 (2003), 177–202.

[41] Daniel Perelman, Sumit Gulwani, Dan Grossman, and Peter Provost. 2014. Test-driven synthesis. In *PLDI*. ACM, 408–418.

[42] M. Rusinowitch, R. Küsters, M. Turuani, and Y. Chevalier. 2003. An NP Decision Procedure for Protocol Insecurity with XOR. In *Logic in Computer Science, Symposium on*. IEEE Computer Society, Los Alamitos, CA, USA, 261.

[43] Guillaume Scerri and Ryan Stanley-Oakes. 2016. Analysis of Key Wrapping APIs: Generic Policies, Computational Security. In *CSF*. IEEE Computer Society, 281–295.

[44] Victor Shoup. 2004. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptol. ePrint Arch.* (2004), 332.

Outline of appendices. We give missing details of the logic's syntax and semantics in Appendix A, and the semantics of our programming language is presented in Appendix B. In Appendix C we develop the coupling method introduced in Section 4.3. In Appendix D, we present our complete proof system for bideduction. Finally, Appendix E provides more details on the implementation of our **crypto** tactic, and shows how the Hash Lock protocol's security can be proved in Squirrel with this tactic.

A LOGIC

We present in more details the indistinguishability logic of [8], including elided definitions, introducing new technical definitions necessary for the rest of the appendices, and adding explanations and examples.

A key aspect of the logic is that the probabilistic semantics of a term is represented as a deterministic function taking random tapes as input. This yields an *eager sampling* semantics for our logic, where no random samplings are performed during the computation of the semantics. Instead, all necessary randomness is drawn in advance, and the semantics retrieve random values from it as needed. Using such a semantics makes all sources of randomness explicit, and allows to easily track the randomness shared between different computations.

Type structures and randomness. Any type structure $\mathbb{M}$ must provide the sampling procedures used to sample values in each type. First, for every base type $\tau_b \in \mathbb{B}$ and η, a type structure $\mathbb{M}$ defines the number $\mathsf{R}_{\mathbb{M},\eta}(\tau_b) \in \mathbb{N}$ of random bits needed to sample a value of type τ_b. Second, $\mathbb{M}$ provides a machine $[\![\tau_b]\!]^{\$}_{\mathbb{M}}$ such that for every η and bitstring w of length $\mathsf{R}_{\mathbb{M},\eta}(\tau_b)$, $[\![\tau_b]\!]^{\$}_{\mathbb{M}}(1^\eta, w)$ computes a value in $[\![\tau_b]\!]^{\eta}_{\mathbb{M}}$ in time polynomial in η.

Term interpretation. For $X \in \mathbb{RV}_{\mathbb{M}}(\tau)$, we define $\mathbb{M}[x \mapsto X]$ as the model which maps x to X and is otherwise identical to $\mathbb{M}$. The interpretation $[\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} \in \mathbb{RV}_{\mathbb{M}}(\tau)$ of any term t of type τ is defined as follows:

$$[\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} \stackrel{\text{def}}{=} \mathbb{M}(x)(\eta)(\rho) \qquad (\text{if } (x:\tau) \in \mathcal{E})$$

$$[\![t\ t']\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} \stackrel{\text{def}}{=} [\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}([\![t']\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})$$

$$[\![\lambda(x:\tau_0).\,t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} \stackrel{\text{def}}{=} \begin{cases} [\![\tau_0]\!]^{\eta}_{\mathbb{M}} & \to\ [\![\tau]\!]^{\eta}_{\mathbb{M}} \\ a & \mapsto\ [\![t]\!]^{\eta,\rho}_{\mathbb{M}[x\mapsto \mathbb{1}^\eta_a]:(\mathcal{E},x:\tau_0)} \end{cases}$$

where $\mathbb{1}^\eta_a$ is the element of $\mathbb{RV}_{\mathbb{M}}(\tau_0)$ such that $\mathbb{1}^\eta_a(\eta)(\rho) = a$ for every ρ and $\mathbb{1}^\eta_a(\eta')(\rho)$ is some irrelevant value for $\eta \neq \eta'$. Note that the parameters η and ρ remain constant throughout the interpretation (an interpretation $[\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$ only depends on interpretations of subterms of t for the same parameters η, ρ) which reflects the eager sampling semantics announced before.

Names. Let $\mathcal{E}$ be an environment and n a name with type $\tau_0 \to \tau_1$ in $\mathcal{E}$. Then, in any model $\mathbb{M} : \mathcal{E}$, we require that there exist a machine w_{n} such that, for every $\eta \in \mathbb{N}$ and $a \in [\![\tau_0]\!]^{\eta}_{\mathbb{M}}$, $w_{\mathsf{n}}(\eta, a, \rho_{\mathsf{h}})$ extracts, in time polynomial in η, $\mathsf{R}_{\mathbb{M},\eta}(\tau_1)$ consecutive random bits from the honest tape ρ_{h}. Furthermore, we require that $w_{\mathsf{n}}(\eta, a, \rho_{\mathsf{h}})$ and $w_{\mathsf{n}'}(\eta, a', \rho_{\mathsf{h}})$ extract disjoint parts of ρ_{h} when either the names n, n' or the indices a, a' differ. Then, the interpretation of the name n is obtained by feeding the random bits extracted by w_{n} to the sampling algorithm $[\![\tau_1]\!]^{\$}_{\mathbb{M}}$ given by the type structure underlying $\mathbb{M}$. Then, we must have that:

$$[\![\mathsf{n}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} \stackrel{\text{def}}{=} \begin{cases} [\![\tau_0]\!]^{\eta}_{\mathbb{M}} & \to\ [\![\tau_1]\!]^{\eta}_{\mathbb{M}} \\ a & \mapsto\ [\![\tau_1]\!]^{\$}_{\mathbb{M}}(\eta, w_{\mathsf{n}}(\eta, a, \rho_{\mathsf{h}})) \end{cases}$$

By construction, if $\mathsf{n}_1 : \tau_1 \to \tau$ and $\mathsf{n}_2 : \tau_2 \to \tau$ are distinct names and $a_1 \in [\![\tau_1]\!]^{\eta}_{\mathbb{M}}, a_2 \in [\![\tau_2]\!]^{\eta}_{\mathbb{M}}$, the random variables $\rho \mapsto [\![\mathsf{n}_1]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(a_1)$ and $\rho \mapsto [\![\mathsf{n}_2]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(a_2)$ are independent and identically distributed, for any η. The same holds for $[\![\mathsf{n}_1]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(a_1)$ and $[\![\mathsf{n}_1]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(a_1')$ whenever $a_1' \neq a_1$. Going further, if t is a term whose free variables are either names other than n_1, or variables x such

that $\mathrm{adv}(x)$ holds, the random variables $[\![t]\!]^{\eta}_{\mathbb{M}:\mathcal{E}}$ and $[\![\mathsf{n}_1]\!]^{\eta}_{\mathbb{M}:\mathcal{E}}$ are also independent: indeed, the semantics of t only depends on the tape ρ_{a} and segments of ρ_{h} disjoint from the segments extracted by n_1.

Global formulas. The formulas of our first-order logic are standard. We call them *global formulas* and decorate their logical connectives with a tilde to distinguish them from local formulas:

$$F ::= \tilde{\bot} \mid F \tilde{\Rightarrow} F \mid \tilde{\forall}(x:\tau).\, F \mid \\ \mathrm{const}(t) \mid \mathrm{adv}(t) \mid [t]_{\mathsf{e}} \mid [t] \mid t_1, \ldots, t_n \sim t'_1, \ldots, t'_n$$

Other connectives and quantifiers ($\tilde{\neg}, \tilde{\vee}, \tilde{\wedge}, \tilde{\exists}$) are defined from $\tilde{\bot}, \tilde{\Rightarrow}, \tilde{\forall}$ as usual. As for terms, we write $\mathcal{E} \vdash F$ when F is well-typed in $\mathcal{E}$ (we omit the typing rules, which are standard).

The semantics of first-order quantifiers and connectives are as usual. The atom $\mathrm{const}(t)$ requires that t is a constant value, independent from both η and ρ, i.e. $\mathbb{M} \models \mathrm{const}(t)$ iff. there exists c such that $[\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = c$ for every η, ρ. We already described the semantics of the $\mathrm{adv}(t)$, $[t]_{\mathsf{e}}$, $[t]$ and $t_1, \ldots, t_n \sim t'_1, \ldots, t'_n$ atoms in Section 3.1.

Example 7. *The global formulas $[\varphi]\ \tilde{\wedge}\ [\varphi]$ and $[\varphi \wedge \varphi]$ are logically equivalent, but this does not hold with disjunctions. Finally, the following axiom scheme is valid, for any terms $\vec{u}, \vec{v}$:*

$$\tilde{\forall} x \tilde{\forall} y.\ [x = y] \tilde{\Rightarrow} (\vec{u} \sim \vec{v}) \tilde{\Rightarrow} (\vec{u}\{x \mapsto y\} \sim \vec{v}\{x \mapsto y\})$$

where, for any terms t, t' and variable x, we let $t\{x \mapsto t'\}$ be the term t in which all occurrences of x have been substituted by t'.

Example 8. *For any $t : \mathsf{bool}$ we have $\mathbb{M} \models [t]_{\mathsf{e}} \tilde{\Rightarrow} [t]$ for any $\mathbb{M}$, i.e. that formula is valid. The converse implication is not valid. Moreover, $[t]$ is logically equivalent to $t \sim \mathrm{true}$.*

B PROGRAM SEMANTICS

We present here the semantics of our expressions and programs.

B.1 Program Random Tapes

To fit with the logic, all the randomness of our programs is sampled eagerly and passed to the program using read-only random tapes. To sample a value of type τ_{b}, we retrieve a vector $w_{\$}$ of $\mathsf{R}_{\mathbb{M},\eta}(\tau_{\mathsf{b}})$ bits from the random tapes, and then use the sampling algorithm $[\![\tau_{\mathsf{b}}]\!]^{\$}_{\mathbb{M}}(\eta, w_{\$})$ provided by the model to obtain a value in $[\![\tau_{\mathsf{b}}]\!]^{\eta}_{\mathbb{M}}$. To simplify the presentation and analysis of the bi-deduction logic in Section 4, we use a different random tape for each usage: we will use a family of random tapes, one for each pair $(\mathsf{T}, \tau_{\mathsf{b}})$ of randomness source (i.e. tag $\mathsf{T} \in \{\mathsf{T_A}, \mathsf{T_G}, \mathsf{T_S}\}$) and base type $\tau_{\mathsf{b}} \in \mathbb{B}$ we are sampling from. However, we only consider bool for $\mathsf{T_A}$ since adversarial randomness is only needed for the adversarial function symbols in $\mathcal{L}_{\mathsf{p}}$.

Definition 8. *A* program random tape $\mathfrak{p}$ *is a family* $(\mathfrak{p}|_l)_{l \in L}$ *of infinite sequences of bits indexed by the set of* labels*:*

$$L \stackrel{\text{def}}{=} \{(\mathsf{T_A}, \mathsf{bool})\} \cup \bigcup_{\tau_b \in \mathbb{B}} \{(\mathsf{T_G}, \tau_b)\} \cup \{(\mathsf{T_S}, \tau_b)\}.$$

For any tag T, we split $\mathfrak{p}|_{T,\tau}$ into blocks of $\mathsf{R}_{\mathbb{M},\eta}(\tau)$ bits, and for any $k \in \mathbb{N}$, we let $\mathfrak{p}|^{\eta,\mathbb{M}}_{T,\tau}[k]$ be the k-th such block. We may omit $\mathbb{M}$ and τ when they are clear from the context.

Finally, we let $\mathfrak{P}$ be the set of all program random tapes.

$$[\mathsf{b}]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu} \stackrel{\text{def}}{=} i \qquad [v]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu} \stackrel{\text{def}}{=} \mu(v) \quad \text{when } v \in \mathcal{X}_{\mathsf{p}}$$

$$[f]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu} \stackrel{\text{def}}{=} [\![f]\!]^{\eta,(\mathfrak{p}|_{\mathsf{T_A},\mathsf{bool}},\rho_{\mathbb{0}})}_{\mathbb{M}:\mathcal{E}} \quad \text{when } f \in \mathcal{L}_{\mathsf{p}}$$

$$[e_1\ e_2]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu} \stackrel{\text{def}}{=} [e_1]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu}([e_2]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu})$$

Figure 7: Semantics of expressions w.r.t. an model $\mathbb{M} : \mathcal{E}$.

$$\mu^{i\ \eta,\mathfrak{p}}_{\mathrm{init}\,\mathbb{M}}(\cdot) \stackrel{\text{def}}{=} \{\mathrm{eta} \mapsto \eta\}$$

$$\mu^{i\ \eta,\mathfrak{p}}_{\mathrm{init}\,\mathbb{M}}(\mathrm{G}) \stackrel{\text{def}}{=} \mu^{i\ \eta,\mathfrak{p}}_{\mathrm{init}\,\mathbb{M}}(\mathrm{decl\_vars_G})$$

$$\mu^{i\ \eta,\mathfrak{p}}_{\mathrm{init}\,\mathbb{M}}(\mathrm{decls}; v \leftarrow e) \stackrel{\text{def}}{=} (\!|v \leftarrow e|\!)^{\eta,\mathfrak{p}}_{\mu} \quad \text{where } \mu = \mu^{i\ \eta,\mathfrak{p}}_{\mathrm{init}\,\mathbb{M}}(\mathrm{decls})$$

Figure 8: Initial memory of a game G w.r.t. $\mathbb{M}$ and side bit i.

B.2 Expression Semantics

We say that a logical environment $\mathcal{E}$ is *compatible* with the set of program variables $\mathcal{X}_{\mathsf{p}}$ and library $\mathcal{L}_{\mathsf{p}}$ if $\mathcal{L}_{\mathsf{p}} \subseteq \mathcal{E}$ and the set of variables defined or declared in $\mathcal{E}$ is disjoint from $\mathcal{X}_{\mathsf{p}}$.

The semantics $[e]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu}$ of an expression e of type τ is a value in $[\![\tau]\!]^{\eta}_{\mathbb{M}}$. This semantics is evaluated relatively to a memory μ, a model $\mathbb{M} : \mathcal{E}$ such that $\mathcal{X}_{\mathsf{p}}$, $\mathcal{L}_{\mathsf{p}}$ and $\mathcal{E}$ are compatible, a security parameter η, a program random tape $\mathfrak{p}$, and a bit $i \in \{0, 1\}$ stating on which side the expression is evaluated. The semantics of expressions, defined in Fig. 7, uses the bit i to interpret the special boolean term b, and the memory μ to evaluate program variables in $\mathcal{X}_{\mathsf{p}}$. Moreover, the semantics of a library function $f \in \mathcal{L}_{\mathsf{p}}$ is:

$$[f]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu} \stackrel{\text{def}}{=} [\![f]\!]^{\eta,(\mathfrak{p}|_{\mathsf{T_A},\mathsf{bool}},\rho_{\mathbb{0}})}_{\mathbb{M}:\mathcal{E}}$$

i.e. the (logical) semantics of f in the model $\mathbb{M}$, using $\mathfrak{p}|_{\mathsf{T_A},\mathsf{bool}}$ as adversarial (logical) random tape, and the all-zero random tape $\rho_{\mathbb{0}}$ as honest random tape — indeed all library function $\mathcal{L}_{\mathsf{p}}$ will be assumed to be adversarial, and therefore do not need *honest* randomness.

We omit $\mathbb{M}$ and i when they are clear from context, and write $[e]^{\eta,\mathfrak{p}}_{\mu}$ instead of $[e]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu}$.

B.3 Initial Memory

The initial memory $\mu^{i\ \eta,\mathfrak{p}}_{\mathrm{init}\,\mathbb{M}}(\mathrm{G})$ of a game G for security parameter η, program random tape $\mathfrak{p}$, model $\mathbb{M}$ and side bit $i \in \{0, 1\}$ is defined in Fig. 8. It is obtained by evaluating the deterministic global variable assignments. Moreover, the value of the security parameter is made available to the game and the simulator through the variable eta. Global random variables are not in this initial memory; they will be sampled during oracle calls.

B.4 Program Semantics

The semantics of a program is parameterized by the game G that the program can interact with, a model $\mathbb{M} : \mathcal{E}$ (such that $\mathcal{X}_{\mathsf{p}}$, $\mathcal{L}_{\mathsf{p}}$ and $\mathcal{E}$ are compatible) used to interpret library function symbols, the side bit $i \in \{0, 1\}$, and the security parameter η. The evaluation $(\!|\mathsf{p}|\!)^{\eta,\mathfrak{p}}_{\mathrm{G},\mathbb{M},i,\mu} \in \mathsf{Mem}_{\mathbb{M},\eta} \cup \{\bot\}$ of a program p in memory μ and

$$(\!|v \leftarrow e|\!)_{\mu}^{\eta,\mathfrak{p}} \stackrel{\text{def}}{=} \mu\left[v \mapsto [e]_{\mu}^{\eta,\mathfrak{p}}\right] \qquad (\!|\textbf{abort}|\!)_{\mu}^{\eta,\mathfrak{p}} \stackrel{\text{def}}{=} \bot \qquad (\!|\textbf{skip}|\!)_{\mu}^{\eta,\mathfrak{p}} \stackrel{\text{def}}{=} \mu$$

$$(\!|\mathsf{p}_0;\mathsf{p}_1|\!)_{\mu}^{\eta,\mathfrak{p}} \stackrel{\text{def}}{=} \begin{cases} (\!|\mathsf{p}_1|\!)_{\mu'}^{\eta,\mathfrak{p}} & \text{if } (\!|\mathsf{p}_0|\!)_{\mu}^{\eta,\mathfrak{p}} = \mu' \\ \bot & \text{if } (\!|\mathsf{p}_0|\!)_{\mu}^{\eta,\mathfrak{p}} = \bot \end{cases}$$

$$(\!|\textbf{if } e \textbf{ then } \mathsf{p}_0 \textbf{ else } \mathsf{p}_1|\!)_{\mu}^{\eta,\mathfrak{p}} \stackrel{\text{def}}{=} \begin{cases} (\!|\mathsf{p}_0|\!)_{\mu}^{\eta,\mathfrak{p}} & \text{if } [e]_{\mu}^{\eta,\mathfrak{p}} = \mathsf{true} \\ (\!|\mathsf{p}_1|\!)_{\mu}^{\eta,\mathfrak{p}} & \text{if } [e]_{\mu}^{\eta,\mathfrak{p}} = \mathsf{false} \end{cases}$$

$$(\!|\textbf{while } e \textbf{ do } \mathsf{p}|\!)_{\mu}^{\eta,\mathfrak{p}} \stackrel{\text{def}}{=} \lim_{n\to\infty} (\!|\textbf{loop}_n|\!)_{\mu}^{\eta,\mathfrak{p}}$$

$$\text{where } \textbf{loop}_n = (\textbf{if } e \textbf{ then } \mathsf{p} \textbf{ else skip})^n; \textbf{if } e \textbf{ then abort else skip}$$

$$(\!|v \xleftarrow{\$} \mathsf{T}[e]|\!)_{\mu}^{\eta,\mathfrak{p}} \stackrel{\text{def}}{=} \mu\left[v \mapsto [\![\tau]\!]_{\mathbb{M}}^{\$}(\eta, \mathfrak{p}|_{(\mathsf{T},\tau)}^{\eta}[k])\right] \text{ where } k = [e]_{\mu}^{\eta,\mathfrak{p}} \text{ and } v \text{ has type } \tau$$

$$(\!|v \leftarrow O_f(\vec{e})[\vec{e_g};\vec{e_l}]|\!)_{\mu}^{\eta,\mathfrak{p}} \stackrel{\text{def}}{=} \text{let } \mu' = \mu\begin{bmatrix} f.\mathrm{args} & \mapsto [\vec{e}]_{\mu}^{\eta,\mathfrak{p}} \\ \mathrm{G.glob}_{\$} & \mapsto \mathfrak{p}|_{\mathsf{T_G}}^{\eta}[\,[\vec{e_g}]_{\mu}^{\eta,\mathfrak{p}}\,] \\ f.\mathrm{loc}_{\$} & \mapsto \mathfrak{p}|_{\mathsf{T_G}}^{\eta}[\,[\vec{e_l}]_{\mu}^{\eta,\mathfrak{p}}\,] \end{bmatrix} \text{ in}$$

$$\text{let } \mu'' = (\!|f.\mathrm{prog}|\!)_{\mu'}^{\eta,\mathfrak{p}} \text{ in}$$

$$\mu''\left[v \mapsto [f.\mathrm{expr}]_{\mu''}^{\eta,\mathfrak{p}}\right]$$

Figure 9: Program semantics w.r.t. a model $\mathbb{M} : \mathcal{E}$, a side $i \in \{0,1\}$ and a game G.

using the program random tape $\mathfrak{p}$ is either the memory obtained by executing p, or $\bot$ if the execution does not terminate. Its definition, given in Fig. 9, is mostly standard; we describe next the treatment of oracle calls and samplings.

If v is a variable of type τ, then the evaluation of the random sampling $v \xleftarrow{\$} \mathsf{T}[e]$ w.r.t. memory μ and program random tape $\mathfrak{p}$ evaluates the integer e as an offset $k \in \mathbb{N}$, retrieves the k-th block of random bits $\mathfrak{p}|_{(\mathsf{T},\tau)}^{\eta}[k]$ from the random tape labeled by (T, τ), and uses it to run the sampling algorithm $[\![\tau]\!]_{\mathbb{M}}^{\$}$ provided by the model $\mathbb{M}$.

To evaluate an oracle call instruction $v \leftarrow O_f(\vec{e})[\vec{e_g};\vec{e_l}]$, we first evaluate the arguments $\vec{e}$, the global randomness offsets $\vec{e_g}$ and the local randomness offsets $\vec{e_l}$, and store the results in, resp., $f.\mathrm{args}$, $\mathrm{G.glob}_{\$}$ and $f.\mathrm{loc}_{\$}$; then, we execute the oracle body $f.\mathrm{prog}$; and finally, we store the result of the evaluation of the return expression $f.\mathrm{expr}$ in v.

B.5 Cost Model

To keep our approach generic and abstract, we assume that our program semantics is endowed with a time-cost model satisfying some standard and expected properties.

More precisely, we assume a cost function C parameterized by the model $\mathbb{M}$ which associates to each program p, security parameter η and memory μ a worst-case execution time $C_{\mathbb{M}}(\mathsf{p}, \eta, \mu) \in \mathbb{N} \cup \{+\infty\}$ which bounds execution times of p for all possible program tapes — this cost must be $+\infty$ if some execution does not terminate. We say that a program p is PTIME w.r.t. $\mathbb{M}$ when $C_{\mathbb{M}}(\mathsf{p}, \eta, \mu)$ is bounded by a polynomial in η and $|\mu|$ (the sum of the sizes of all values stored in μ). We will assume only a few basic properties of this cost model:

- all expressions are PTIME, which is reasonable as sampling procedures provided by the model are PTIME, and since library functions are assumed to be adversarial;
- the memory after executing a PTIME program is of polynomial size in η and the size of the initial memory;
- an oracle call is PTIME, which is both a constraint on the cost model and the game;
- if both p and q are PTIME programs, then so is $(p; q)$;
- **while** $l \neq []$ **do** (p; $l \leftarrow \mathrm{tail}\ l$) is PTIME provided that p is a PTIME program that does not modify variable l, in all models where tail induces a well-founded ordering on the semantic values of type list.

B.6 Adversaries

An adversary against G (or G-adversary) with respect to a model $\mathbb{M} : \mathcal{E}$ and a security parameter η is a program which may only call the oracles of G, respecting their type. Moreover, an adversary must not read the special side constant b, and must not read or write the game variables. Finally, the program must properly use random samplings, in any possible execution in $\mathbb{M}$, with the security parameter η:

- We forbid the adversary from directly sampling from the $\mathsf{T_G}$-labeled random tapes, which are reserved for the game's random samplings.
- We require that local offsets in oracle calls are fresh: an integer used as a local offset may not be used anywhere else as an offset, in this oracle or in a past or future call.
- We require that global offsets are consistent across all oracle calls: each of the game's global samplings must correspond to a unique global offset.

C PROBABILISTIC COUPLINGS AND BI-DEDUCTION

In this appendix we go back to Section 4.3, where we intuitively introduced the notion of well-formedness for constraint systems,

coming from the need to lift semantical equalities to probabilistic equalities. We show a counter-example illustrating the need for the well-formedness condition, then define formally this condition, and prove Lemma 1.

Example 9. *Consider a name* $\mathsf{n} : \mathsf{unit} \to \mathsf{bool}$ *and let* $C = \{c_0, c_1\}$ *with:*

$$c_0 = (\emptyset, \mathsf{n}, \langle\rangle, \mathsf{T}_S, \mathsf{n}\ \langle\rangle = 0)$$
$$c_1 = (\emptyset, \mathsf{n}, \langle\rangle, \mathsf{T}_G^{\mathsf{loc}}, \mathsf{n}\ \langle\rangle = 1)$$

In words, $\mathsf{n}\ \langle\rangle$ *must be seen as a simulator name when it is 0, and a local sampling of the game when it is 1. But, to know in which case we are, we must already have sampled* $\mathsf{n}\ \langle\rangle$*!*

Let us show that a coupling cannot be included in $\mathcal{R}^{\eta}_{C,\mathbb{M}}$*. First observe that* $\rho\ \mathcal{R}^{\eta}_{C,\mathbb{M}}\ \rho_a$ *imposes that* ρ_a *is a prefix of* $\mathfrak{p}[\mathsf{T_A}, \mathsf{bool}]$ *and:*

- *either* $[\![\mathsf{n}\ \langle\rangle]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = 0$ *and* $\mathfrak{p}|^{\eta}_{\mathsf{T}_S}[O_{\mathbb{M},\eta}(\mathsf{n}, \langle\rangle)] = 0$*;*
- *or* $[\![\mathsf{n}\ \langle\rangle]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = 1$ *and* $\mathfrak{p}|^{\eta}_{\mathsf{T}^{\mathsf{loc}}_G}[O_{\mathbb{M},\eta}(\mathsf{n}, \langle\rangle)] = 1$*.*

Less formally, the logical tape must coincide with the simulator tape on $\mathsf{n}\ \langle\rangle$ *when this sampling is zero; otherwise it must coincide with the local sampling tape for that name. Thus, the program tape* ρ_a *such that* $\mathfrak{p}|^{\eta}_{\mathsf{T}_S}[O_{\mathbb{M},\eta}(\mathsf{n}, \langle\rangle)] = 0$ *and* $\mathfrak{p}|^{\eta}_{\mathsf{T}^{\mathsf{loc}}_G}[O_{\mathbb{M},\eta}(\mathsf{n}, \langle\rangle)] = 1$ *is not related to any logical tape in* $\mathcal{R}^{\eta}_{C,\mathbb{M}}$ *— for any* ρ*, we do not have* $\rho\ \mathcal{R}^{\eta}_{C,\mathbb{M}}\ \rho_a$*. Hence the right marginal of a coupling included in* $\mathcal{R}^{\eta}_{C,\mathbb{M}}$ *would never sample such tapes. This missing set of tapes has non-zero measure (in fact it has measure* $\frac{1}{4}$*) hence the right marginal of our coupling would not coincide with the standard distribution over program tapes, which is a contradiction.*

C.1 Preliminaries: Probability Theory

We first recall some standard definitions from measure and probability theory.

Definitions. For any set $\mathbb{S}$, we let $\mathcal{P}(\mathbb{S})$ be the *power-set* of $\mathbb{S}$. A *σ-algebra* $\mathcal{F}$ over a set $\mathbb{S}$ is a non-empty subset of $\mathcal{P}(\mathbb{S})$ closed under: i) complement; and ii), countable union and intersection. An element E of a σ-algebra is called an *event*. A *measurable space* $(\mathbb{S}, \mathcal{F})$ is a set $\mathbb{S}$ equipped with a σ-algebra $\mathcal{F}$. A *measure space* $(\mathbb{S}, \mathcal{F}, \mu)$ is a measurable set $(\mathbb{S}, \mathcal{F})$ together with a function $\mu : \mathcal{F} \to [0; 1]$ — called a *measure* — such that i) $\mu(\emptyset) = 0$; ii) μ is non-negative (i.e. $\forall E \in \mathcal{F}, \mu(E) \geq 0$); iii) μ is σ-additive, i.e. for any countable sequences $(E_i)_{i\in\mathbb{N}}$ of disjoint elements of $\mathcal{F}$, $\mu(\bigcup_i E_i) = \sum_i \mu(E_i)$. A *probability space* $(\mathbb{S}, \mathcal{F}, \mu)$ is a measure space of total mass is 1, i.e. $\mu(\mathbb{S}) = 1$. A *distribution* D over a measurable space $(\mathbb{S}, \mathcal{F})$ is a function such that $(\mathbb{S}, \mathcal{F}, D)$ is a probability space. Two distributions D_1 and D_2 over $(\mathbb{S}, \mathcal{F})$ are said to be of the *same law* if $D_1(E) = D_2(E)$ for any $E \in \mathcal{F}$. Finally, a *random variable* $X : \Omega \to \mathbb{S}$ from a probability space $(\Omega, \mathcal{F}_\Omega, \mu_\Omega)$ to a measurable space $(\mathbb{S}, \mathcal{F}_\mathbb{S})$ is any function such that $\forall E \in \mathcal{F}_\mathbb{S}, X^{-1}(E) \in \mathcal{F}_\Omega$.

Notations. If $(\mathbb{S}, \mathcal{F}, \mu)$ is a measure space and E an event of $\mathcal{F}$, then the probability $\Pr(E)$ of E is simply $\mu(E)$. Similarly, if D is a distribution over $(\mathbb{S}, \mathcal{F})$ and E an event of $\mathcal{F}$, then $\Pr(D \in E) \stackrel{\text{def}}{=} D(E)$. If X is a random variable from the probability space $(\Omega, \mathcal{F}_\Omega, \mu_\Omega)$ to $(\mathbb{S}, \mathcal{F}_\mathbb{S})$ and E an event of $\mathcal{F}_\mathbb{S}$, then $\Pr(X \in E) \stackrel{\text{def}}{=} \mu(X^{-1}(E))$.

Distributions as programs. We will describe some distributions using programs written in pseudo-code, e.g. if D is a distribution, then the program $\mathsf{x} \xleftarrow{\$} D; \mathsf{y} \xleftarrow{\$} D; \mathbf{return}\ (\mathsf{x}, \mathsf{x} + \mathsf{y})$ defines a distribution over pair of values. Given a program p, we write $\Pr_p(E)$ the probability of event E w.r.t. the distribution defined by p.

π and λ systems. Let $\mathbb{S}$ be a set and $X \subseteq \mathcal{P}(\mathbb{S})$, then:

- $\sigma(X)$ is the smallest σ-algebra containing X — we say that X generates $\sigma(X)$.
- X is a *π-system* if X is closed under finite intersections.
- X is a *λ-system* if $\emptyset \in X$ and X is closed under complement and countable disjoint unions.

We recall the following standard result:

Proposition 2 (Dynkin (π, λ)-Theorem)**.** *Let P be a π-system and L a λ-system. If $P \subseteq L$ then $\sigma(P) \subseteq L$.*

To show that two distribution coincide, it is sufficient to show that they coincide on a generating π-system B.

Proposition 3. *Let $(\mathbb{S}, \mathcal{F})$ be a measurable set and D_1, D_2 be two distributions over $\mathbb{S}$. Let B by a π-system such that $\sigma(B) = \mathcal{F}$. If D_1 and D_2 agree on B then D_1 and D_2 agree on $\mathcal{F}$, i.e.*

$$\text{if } \forall E \in B, D_1(E) = D_2(E) \text{ then } \forall E \in \mathcal{F}, D_1(E) = D_2(E)$$

Proof. Let $L \stackrel{\text{def}}{=} \{E \in \mathcal{F} \mid D_1(E) = D_2(E)\}$. We can check that L is a λ-system. By hypothesis, $B \subseteq L$. Hence, by Dynkin (π, λ)-theorem, $\sigma(B) \subseteq L$, which, since B generates $\mathcal{F}$, means that $\mathcal{F} \subseteq L$. Moreover, we trivially have from the definition of L that $L \subseteq \mathcal{F}$. Hence $\mathcal{F} = L$, and thus that D_1 and D_2 coincides on $\mathcal{F}$. □

C.2 Couplings and Lifting Lemma

Recall that, in section Section 4.3, in order to be able to *lift* equalities over tapes in $\mathcal{R}^{\eta}_{C,\mathbb{M}}$ to equalities over probabilities, we relied on the standard notion of a probabilistic coupling and lifting (as in [13]).In this section, we give the definition of probabilistic coupling, before defining containment and a general lifting lemma.

Definition 9 (Probabilistic coupling)**.** *Let $(\mathbb{S}_1, \mathcal{F}_1, \mu_1)$ and $(\mathbb{S}_2, \mathcal{F}_2, \mu_2)$ be two probabilistic spaces.A coupling $\mathbb{C}$ of μ_1 and μ_2, written $\mathbb{C} : \mu_1 \bowtie \mu_2$, is a random variable $\mathbb{C} : \Omega \to \mathbb{S}_1 \times \mathbb{S}_2$ from some probabilistic space Ω to $\mathbb{S}_1 \times \mathbb{S}_2$ such that:*

- *μ_1 and $\mathbb{C}$'s* left marginal *follow the same law, i.e.:*

$$\forall E_1 \in \mathcal{F}_1.\ \Pr_{\mu_1}(E_1) = \Pr(C \in E_1 \times \mathbb{S}_1).$$

- *similarly, μ_2 and $\mathbb{C}$'s* right marginal *follow the same law.*

The coupling we build will be contained in the relation $\mathcal{R}^{\eta}_{C,\mathbb{M}}$, ensuring that only related tapes are coupled.

Definition 10 (Probabilistic containement)**.** *Let $(\mathbb{S}, \mathcal{F}, \mu)$ be a probabilistic space and $E \in \mathcal{F}$ an event. We say that the measure μ is* contained *in E, when for all $F \in \mathcal{F}$, $\mu(F) = \mu(F \cap E)$.*

The following lemma allows to lift an equality over elements related by a relation R to a equality over probabilities, as long as there exists a probabilistic coupling contained in R.

Lemma 2. *Let $(\mathbb{S}_1, \mathcal{F}_1, \mu_1)$ and $(\mathbb{S}_2, \mathcal{F}_2, \mu_2)$ be two probabilistic spaces, $R \subseteq \mathbb{S}_1 \times \mathbb{S}_2$ a relation between $\mathbb{S}_1$ and $\mathbb{S}_1$ and $E_1 \in \mathcal{F}_1$ and $E_2 \in \mathcal{F}_2$ be events such that:*

$$\text{for all } x\ R\ y,\ \ x \in E_1 \text{ iff. } y \in E_2. \tag{7}$$

Then $\Pr_{\mu_1}(E_1) = \Pr_{\mu_2}(E_2)$ if there exists a coupling $\mu : \mu_1 \bowtie \mu_2$ contained in R.

Proof. First, notice that by Eq. (7):

$$(E_1 \times \mathbb{S}_2) \cap R = (E_1 \times E_2) \cap R \tag{8}$$

and

$$(\mathbb{S}_2 \times E_2) \cap R = (E_1 \times E_2) \cap R. \tag{9}$$

Now, let $\mu : \mu_1 \bowtie \mu_2$ be a coupling contained in R. Then:

$$\begin{aligned}
\Pr_{\mu_1}(E_1) &= \Pr_{\mu}(E_1 \times \mathbb{S}_2) && \text{(left marginal property)}\\
&= \Pr_{\mu}((E_1 \times \mathbb{S}_2) \cap R) && \text{(by containement)}\\
&= \Pr_{\mu}((E_1 \times E_2) \cap R) && \text{(by Eq. (8))}\\
&= \Pr_{\mu}((\mathbb{S}_1 \times E_2) \cap R) && \text{(by Eq. (9))}\\
&= \Pr_{\mu}(\mathbb{S}_1 \times E_2) && \text{(by containement)}\\
&= \Pr_{\mu_2}(E_2) && \text{(right marginal property)}
\end{aligned}$$

which concludes this proof. □

C.3 Well-Formedness of Constraint Systems

The goal of this section is to define the notion of well-formedness of a constraint system used in Lemma 1.

Doing so requires us to first introduce what are constraint instances.

Definition 11 (Constraint instance). *Let $\vec{\alpha} = (\alpha_0, \ldots, \alpha_j)$ be a sequence of variables of type $\vec{\tau} = \tau_0, \ldots, \tau_j$. An instance of a constraint $c = (\vec{\alpha}, \mathsf{n}, t, T, f)$ w.r.t. a type structure $\mathbb{M}_0$ and $\eta \in \mathbb{N}$ is an element $(\vec{a}, c)$ where $\vec{a} = (a_0, \ldots, a_j)$ and for any $i \in \{0, \ldots, j\}$, $a_i \in [\![\tau_i]\!]^{\eta}_{\mathbb{M}_0}$.*

Given a model $\mathbb{M}$ and a random tape ρ, we can interpret a constraint instance as a multi-set in a similar way to what we did with constraints:

$$\mathcal{N}^{\eta,\rho}_{(\vec{a},(\vec{\alpha},\mathsf{n},t,T,f)),\mathbb{M}} \stackrel{\text{def}}{=} \left\{ \langle \mathsf{n}, [\![t]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha}\mapsto \mathbb{1}^{\eta}_{\vec{a}}]}, T\rangle \;\middle|\; [\![f]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha}\mapsto \mathbb{1}^{\eta}_{\vec{a}}]} = \mathsf{true} \right\}$$

We lift this to any sequence l_C of constraint instances as follows:

$$\mathcal{N}^{\eta,\rho}_{l_C,\mathbb{M}} \stackrel{\text{def}}{=} \bigcup_{(\vec{a},c)\in l_C} \mathcal{N}^{\eta,\rho}_{(\vec{a},c),\mathbb{M}}$$

where, in the equation above, $\bigcup$ must be understood as multi-set union in the equation above.

We are now ready to explain what is a well-formed constraint system. Roughly, a constraint system C is well-formed if there exists an ordering $c_1, \ldots, c_n$ of the concrete instances it represents that verifies the property that for any i, the instance $c_i = (\vec{a}, (\vec{\alpha}, \mathsf{n}, t, T, f))$ is such that the index t and condition f can be computed using only the names defined by the previous constraint instances $c_1, \ldots, c_{j-1}$.

Definition 12 (Restriction of a random tape). *The restriction $\rho_{|\mathbb{M},\eta,l_C}$ of a random tape ρ by a sequence of constraint instances l_C w.r.t. a model $\mathbb{M}$ and $\eta \in \mathbb{N}$ is the random tape obtained from ρ by zeroing all random bits that corresponds to names that are **not** in $\mathcal{N}^{\eta,\rho}_{l_C,\mathbb{M}}$.*

Definition 13 (Well-formedness of constraint instances). *A finite sequence $l_C = (c_1, \ldots, c_K)$ of constraint instances is well-formed w.r.t. a model $\mathbb{M} : \mathcal{E}$ and $\eta \in \mathbb{N}$ relatively to the terms $\vec{u}$ when for any $k \le K$, if $c_k = (\vec{a}, (\vec{\alpha}, n, t, T, f))$ then there exists a function g such that*

$$g(\rho_{|\mathbb{M},\eta,l_C^k}, [\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = [\![(t \mid f)]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha}\mapsto \mathbb{1}^{\eta}_{\vec{a}}]:(\mathcal{E},\vec{\alpha})}$$

where $l_C^k = (c_0, \ldots, c_{k-1})$.

We can now define the well-formedness of a constraint system.

Definition 14 (Well-formedness of constraint systems). *A constraint system C is well-formed w.r.t. a model $\mathbb{M}$ and $\eta \in \mathbb{N}$ relatively to a vector of input terms $\vec{u}$ when there exists a sequence l_C of constraints instances such that for any tape ρ, $\mathcal{N}^{\eta,\rho}_{C,\mathbb{M}} = \mathcal{N}^{\eta,\rho}_{l_C,\mathbb{M}}$, and l_C is well-formed w.r.t. $\mathbb{M}, \eta$ relatively to $\vec{u}$.*

In that case, we say that l_C witnesses the well-formedness of C.

We write $\mathcal{E}, \Theta \models_{\mathsf{WF}(\vec{u})} C$ if C is well-formed w.r.t. $\mathbb{M}, \eta$ relatively to $\vec{u}$ for any η and any $\mathbb{M} : \mathcal{E}$ such that $\mathcal{E}, \Theta \models \mathbb{M}$. For pairs $C_\# = \#(C_0; C_1)$ of constraint systems, $\mathcal{E}, \Theta \models_{\mathsf{WF}(\vec{u_\#})} C_\#$ stands for well-formedness of both C_0 relatively to $\vec{u_0}$ and C_1 relatively to $\vec{u_1}$. The fact that the notion is relatively to term comes from the need to compose the well-formedness of constraints system in the way we do for adversaries in the bi-deduction inference rules later on, so morally each constraints systems is allowed access to inputs, just like adversaries. Finally, we say that a constraints system is well-formed if it well-formed relatively to the empty sequence of terms.

Outline. The following two sections of this appendix aims at proving Lemma 1, i.e. that a probabilistic coupling contained in $\mathcal{R}^{\eta}_{C,\mathbb{M}}$ can be constructed from any well-formed and valid constraint systems C. First, we prove a preliminary result showing how to build a coupling between two distributions over arrays of independent and identically distributed (i.i.d. for short) values in Appendix C.4, and we then use this result to prove Lemma 1 in Appendix C.5.

C.4 Couplings Arrays

We prove some preliminary results showing how to build couplings of arrays of values.

I.i.d. Sampling of arrays. Let $\mathbb{I}$ be a finite set, and let $D_{\mathbb{S}}$ be a fixed but arbitrary distribution over some measurable space $(\mathbb{S}, \mathcal{F})$. We identify the set $\mathbb{S}^{\mathbb{I}}$ with arrays indexed by $\mathbb{I}$ of values in $\mathbb{S}$.

Definition 15. *We let $D^{\mathbb{I}}_{\mathbb{S}}$ be the distribution over $\mathbb{S}^{\mathbb{I}}$ (equipped with the product σ-algebra) where all cells are* independently *sampled according to $D_{\mathbb{S}}$, i.e. the distribution defined by the program (in pseudo-code):*

$$\begin{array}{l}
a \leftarrow [\bot \textit{ for } _ \in \mathbb{I}];\\
\textbf{for } (j \in \mathbb{I}) \textbf{ do } \{\, a[j] \xleftarrow{\$} D_{\mathbb{S}};\, \}\\
\textbf{return } a;
\end{array} \tag{10}$$

where $\bot$ is a special element (s.t. $\bot \notin \mathbb{S}$) used to denote a cell that is yet to be sampled.

Proposition 4. *Let $\mathbb{I}$ be a finite set, and p be any program of the form:*

$$\begin{array}{l}\mathsf{a} \leftarrow [\bot \text{ for } \_ \in \mathbb{I}];\\ \mathsf{s} \leftarrow \mathsf{s}_{\mathsf{init}};\\ \textbf{for } (\_ \in |\mathbb{I}|) \textbf{ do } \{\, i \leftarrow \mathsf{f}(\mathsf{s});\ \mathsf{a}[i] \xleftarrow{\$} D_{\mathbb{S}};\ \mathsf{s} \leftarrow \mathsf{g}(\mathsf{s},\mathsf{a});\ \}\\ \textbf{return } \mathsf{a};\end{array} \tag{11}$$

where s_{init}, f and g are arbitrary mathematical deterministic functions such that at the end of the execution of the above program, all cells in $\mathbb{I}$ are sampled.

Then p defines a distribution over $\mathbb{S}^{\mathbb{I}}$ of law $D_{\mathbb{S}}^{\mathbb{I}}$.

Proof. Let $n = |\mathbb{I}|$ and $E_1, \dots, E_n \in \mathcal{F}$ be events of $(\mathbb{S}, \mathcal{F})$. First, let us prove that:

$$\Pr_p(\mathsf{a} \in \textstyle\prod_i E_i) = \Pr_{p_0}(\mathsf{a} \in \prod_i E_i) \tag{12}$$

where p_0 is the program sampling the array in an i.i.d. fashion as described in Eq. (10) (hence $\Pr_{p_0}(\mathsf{a} \in \prod_i E_i) = \prod_i \Pr(D_{\mathbb{S}} \in E_i)$). We start be splitting the sum:

$$\Pr_p(\mathsf{a} \in \textstyle\prod_i E_i) \;=\; \sum_\sigma \Pr_p((\mathsf{a} \in \prod_i E_i) \mid A_\sigma) \cdot \Pr_p(A_\sigma)$$

where the sum is over all permutations of $\{1, \dots, n\}$ and A_σ is the event: "p sampled values in the array in the order σ". Conditioned by A_σ, the probability that p samples an array in $\prod_i E_i$ is the probability that the program:

$$\begin{array}{l}\mathsf{a} \leftarrow [\bot \text{ for } \_ \in \mathbb{I}];\\ \textbf{for } (j \in \mathbb{I}) \textbf{ do } \{\, \mathsf{a}[\sigma(j)] \xleftarrow{\$} D_{\mathbb{S}};\ \}\\ \textbf{return } \mathsf{a};\end{array}$$

samples an array in $\prod_i E_i$, i.e. $\prod_i \Pr(D_S \in E_{\sigma^{-1}(i)})$. Hence:

$$\begin{aligned} &\textstyle\sum_\sigma \Pr_p((\mathsf{a} \in \prod_i E_i) \mid A_\sigma) \cdot \Pr_p(A_\sigma)\\ =\ &\textstyle\sum_\sigma \prod_i \Pr(D_S \in E_{\sigma^{-1}(i)}) \cdot \Pr_p(A_\sigma)\\ =\ &\textstyle\prod_i \Pr(D_S \in E_i) \cdot \sum_\sigma \Pr_p(A_\sigma)\\ =\ &\textstyle\prod_i \Pr(D_S \in E_i)\end{aligned}$$

This concludes the proof of Eq. (12).

To finish the proof, we must show that $\Pr_p(\mathsf{a} \in E) = \Pr_{p_0}(\mathsf{a} \in E)$ for any event E in the product σ-algebra $\prod_{1 \le i \le n} \mathcal{F}$. Let B be the set:

$$B \stackrel{\text{def}}{=} \{E_1 \times \cdots \times E_n \mid E_1, \dots, E_n \in \mathcal{F}\}$$

We know that p and $D_{\mathbb{S}}^{\mathbb{I}}$ coincide on B (by Eq. (12)). Moreover, we can check that B is a π-system. By Proposition 3, p and $D_{\mathbb{S}}^{\mathbb{I}}$ agree on the σ-algebra generated by B, which is the product σ-algebra over $\mathbb{S}^{\mathbb{I}}$. Consequently, p is of law $D_{\mathbb{S}}^{\mathbb{I}}$. □

Couplings i.i.d. arrays from selection functions. Let $\mathbb{I}_1$ and $\mathbb{I}_2$ be two finite sets, and let $D_{\mathbb{S}}$ be fixed by arbitrary distribution over a measurable space $(\mathbb{S}, \mathcal{F})$ (the sample space).

Assume that we have a function select such that, for any two partially sampled arrays $\mathsf{a}_1 : \mathbb{I}_1 \to \mathbb{S} \cup \{\bot\}$ and $\mathsf{a}_2 : \mathbb{I}_2 \to \mathbb{S} \cup \{\bot\}$, $(\mathsf{select}\ \mathsf{a}_1\ \mathsf{a}_2)$ either select a pair of $\bot$-valued indices of a_1 and a_2, or return a special value done. More precisely:

$$\begin{array}{c}\forall \mathsf{a}_1, \mathsf{a}_2.\ \mathsf{select}\ \mathsf{a}_1\ \mathsf{a}_2 \in (\mathbb{I}_1 \times \mathbb{I}_2) \cup \{\mathsf{done}\}\\ \text{and } \mathsf{select}\ \mathsf{a}_1\ \mathsf{a}_2 = (i_1, i_2) \Rightarrow \mathsf{a}_1[i_1] = \bot \wedge \mathsf{a}_2[i_2] = \bot\end{array} \tag{13}$$

Let $p_c(\mathsf{select})$ be the distribution over $D_{\mathbb{S}}^{\mathbb{I}_1} \times D_{\mathbb{S}}^{\mathbb{I}_2}$ defined by the program (in pseudo-code):

$$\begin{array}{l}\mathsf{a}_1 \leftarrow [\bot \text{ for } \_ \in \mathbb{I}_1];\\ \mathsf{a}_2 \leftarrow [\bot \text{ for } \_ \in \mathbb{I}_2];\\ \textbf{while } (\mathsf{select}\ \mathsf{a}_1\ \mathsf{a}_2 \neq \mathsf{done}) \textbf{ do } \{\\ \quad (i_1, i_2) \leftarrow \mathsf{select}\ \mathsf{a}_1\ \mathsf{a}_2;\\ \quad \mathsf{v} \xleftarrow{\$} D_{\mathbb{S}};\\ \quad \mathsf{a}_1[i_1] \xleftarrow{\$} \mathsf{v};\\ \quad \mathsf{a}_2[i_2] \xleftarrow{\$} \mathsf{v};\\ \}\\ \textbf{for } (i \in \mathbb{I}_1) \textbf{ do } \{\, \textbf{if } (\mathsf{a}_1[i] = \bot) \textbf{ then } \mathsf{a}_1[i] \xleftarrow{\$} D_{\mathbb{S}};\ \textbf{else skip}\ \}\\ \textbf{for } (i \in \mathbb{I}_2) \textbf{ do } \{\, \textbf{if } (\mathsf{a}_2[i] = \bot) \textbf{ then } \mathsf{a}_2[i] \xleftarrow{\$} D_{\mathbb{S}};\ \textbf{else skip}\ \}\\ \textbf{return } (\mathsf{a}_1, \mathsf{a}_2);\end{array}$$

Proposition 5. *For any selection function* select *satisfying Eq.* (13)*, we have that:*

$$p_c(\mathsf{select}) : D_{\mathbb{S}}^{\mathbb{I}_1} \bowtie D_{\mathbb{S}}^{\mathbb{I}_2}.$$

Proof. It is clear that $p_c(\mathsf{select})$'s left marginal follows the same distribution as:

$$\begin{array}{l}\mathsf{a}_1 \leftarrow [\bot \text{ for } \_ \in \mathbb{I}_1];\\ \mathsf{a}_2 \leftarrow [\bot \text{ for } \_ \in \mathbb{I}_2];\\ \textbf{while } (\mathsf{select}\ \mathsf{a}_1\ \mathsf{a}_2 \neq \mathsf{done}) \textbf{ do } \{\\ \quad (i_1, i_2) \leftarrow \mathsf{select}\ \mathsf{a}_1\ \mathsf{a}_2;\\ \quad \mathsf{a}_1[i_1] \xleftarrow{\$} D_{\mathbb{S}};\\ \quad \mathsf{a}_2[i_2] \leftarrow \mathsf{a}_1[i_1];\\ \}\\ \textbf{for } (i \in \mathbb{I}_1) \textbf{ do } \{\, \textbf{if } (\mathsf{a}_1[i] = \bot) \textbf{ then } \mathsf{a}_1[i] \xleftarrow{\$} D_{\mathbb{S}};\ \textbf{else skip}\ \}\\ \textbf{return } \mathsf{a}_1;\end{array}$$

This program samples all the cells of a_1 independently according to the distribution $D_{\mathbb{S}}$, in some particular order. By Proposition 4, we know that the order in which we sample cells does not matter, and that the distribution defined this program is of law $D_{\mathbb{S}}^{\mathbb{I}_1}$.

Repeating the same reasoning on the right, we get that $p_c(\mathsf{select})$'s right marginal follows the distribution $D_{\mathbb{S}}^{\mathbb{I}_2}$, which concludes this proof. □

C.5 Constructing a Coupling Contained in $\mathcal{R}^{\eta}_{C,\mathbb{M}}$

We now recall and prove Lemma 1.

Lemma 1. *Let C be a well-formed constraint system w.r.t. $\mathbb{M}, \eta$ such that $\mathbb{M} \models \mathrm{Valid}(C)$. Then, there exists a coupling $\mathbb{C} : \mathbb{T}_{\mathbb{M},\eta} \bowtie \mathfrak{P}$ contained in $\mathcal{R}^{\eta}_{C,\mathbb{M}}$.*

Proof. Let $\mathbb{M}$ be a model, η a value of the security parameter and C a constraint system such that C is both valid and well-formed w.r.t. $\mathbb{M}$. We are going to build, for any $\eta \in \mathbb{N}$, a coupling that is contained in $\mathcal{R}^{\eta}_{C,\mathbb{M}}$.

We use the framework of Proposition 5 for building couplings. We instantiate it such that a_1 represents a (partially defined) logical tape, which will be noted ρ, and a_2 represents the relevant finite portion of a partial computational tape, noted $\mathfrak{p}$. Given a partial logical tape ρ, mapping each type and index in $\mathsf{R}_{\mathbb{M},\eta}(\tau)$ to a value in $\{0, 1, \bot\}$, we say that a term t is well-defined w.r.t. ρ when $[\![t]\!]^{\eta,\rho_1}_{\mathbb{M}:\mathcal{E}} = [\![t]\!]^{\eta,\rho_2}_{\mathbb{M}:\mathcal{E}}$ for all tapes ρ_1 and ρ_2 that coincide with ρ where it is defined.

When it is the case, we allow ourselves to simply write $[\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$ for this unique value.

We now describe the selection function (select ρ $\mathfrak{p}$) with which we instantiate the framework. Let l_C be a sequence of instances witnessing the well-formedness of C w.r.t. $\mathbb{M}, \eta$. At each iteration, the function select chooses, if it exists, the smallest integer k, such that the k^{th} element in l_C is a constraint instance $(\vec{a}, c)$ with $c = (\vec{\alpha}, \mathsf{n}, t, T, f) \in C$ such that:

(a) both $[\![f]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha}\mapsto\vec{a}]:\mathcal{E},\vec{\alpha}}$ and $[\![t]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha}\mapsto\vec{a}]:\mathcal{E},\vec{\alpha}}$ are defined w.r.t. ρ, and the former is true;
(b) ρ still contains $\bot$ in the segment corresponding to name n and index $[\![t]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha}\mapsto\vec{a}]:\mathcal{E},\vec{\alpha}}$;
(c) $\mathfrak{p}$ still contains $\bot$ in the segment corresponding to name n, index $[\![t]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha}\mapsto\vec{a}]:\mathcal{E},\vec{\alpha}}$ and tag T,

and returns the corresponding indices in the logical and computational tapes. Otherwise, it returns done.

It should be noted that select does not simply consider offsets in the order prescribed by l_C. To explain why this is necessary, consider two equivalent constraint instances $(\vec{a}, c)$ and $(\vec{a'}, c')$, i.e. such that they both satisfy (a) and their indices are the same:

$$[\![t]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha}\mapsto\vec{a}]:\mathcal{E},\vec{\alpha}} = [\![t']\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha}\mapsto\vec{a'}]:\mathcal{E},\vec{\alpha}}.$$

Then, $(\vec{a}, c)$ and $(\vec{a'}, c')$ refers to the same offsets, and the function select should not return twice the same offsets.

We now show that our coupling is contained in $\mathcal{R}^{\eta}_{C,\mathbb{M}}$. To do so, consider an arbitrary run of $p_c(\text{select})$. We say that an instance $(\vec{a}, c)$ is addressed at some point in this run if satisfies (a) but neither (b) nor (c). Once an instance is addressed, the value of the corresponding name will have been set in the tapes. Note, though that an instance needs not be selected to be addressed: it suffices that an equivalent constraint instances is selected.

We observe that, at every step of our run, and for every instance for which condition (a) holds, conditions (b) and (c) are equivalent. Indeed, if only one kind of tape is defined for our name, it must have been set due to the previous selection of another constraint instance, but validity imposes that distinct instances address distinct names.

Then, we note that, for every instance $(\vec{a}, c)$ in the sequence l_C, if all instances preceding $(\vec{a}, c)$ in l_C have been addressed then condition (a) holds. This is a consequence of well-foundedness. Indeed, let $c = (\vec{\alpha}, \mathsf{n}, t, T, f)$ and $l' = \big((\vec{a}_0, c_0), \ldots, (\vec{a}_k, c_k)\big)$ be the instances strictly before the position of the instance $(\vec{a}, c)$ we consider in l_C. We have that for any $\hat{\rho}$, the semantics of $(t \mid f)$ is a function of $\hat{\rho}_{|\mathbb{M},\eta,l'}$. We assumed that all the instances of l' have been addressed. Then for any tapes $\hat{\rho}$ and $\tilde{\rho}$ that coincide with the partial tape ρ, we have that $\hat{\rho}_{|\mathbb{M},\eta,l'} = \tilde{\rho}_{|\mathbb{M},\eta,l'}$, and thus $(t \mid f)$ is well-defined w.r.t. ρ.

To conclude, every instance in l_C will eventually be addressed. Hence, for any $(n, v, \mathsf{T}) \in \mathcal{N}^{\eta,\rho}_{l_C,\mathbb{M}} = \mathcal{N}^{\eta,\rho}_{C,\mathbb{M}}$, we have $[\![n]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(v) = \mathfrak{p}|^{\eta}_{\mathsf{T}}[O_{\mathbb{M},\eta}(n, v)]$. The rest of $\mathcal{R}^{\eta}_{C,\mathbb{M}}$, concerning ρ_a and $\mathfrak{p}[\mathsf{T}_S, \mathrm{bool}]$ is obvious. □

D PROOF SYSTEM

In this section we present the full proof system, which we only sketched in the body.

D.1 Type Tagging and Restrictions

We will need to restrict our attention to type structures satisfying some assumptions. Recall that each proof system rule is proved by providing a simulator, which must be a *polynomial-time* adversary. Besides, to bi-deduce some terms, one might need a simulator with a while-loop, e.g. to compute a function graph. To ensure such simulator still run in polynomial time, one need some restrictions on the types of the loop iterator. These restrictions are presented in this subsection.

For each base types, we use a simple tagging mechanism: we assume that each base type comes with a (possibly empty) set of tags constraining the possible interpretations of the type in type structures. We use the tag finite on a type τ to restrict to structures where $[\![\tau]\!]^{\eta}_{\mathbb{M}}$ is finite for all η, fixed to impose that $[\![\tau]\!]^{\eta}_{\mathbb{M}}$ does not depend on η, and enum to require that there exists a machine $\mathcal{M}^{\tau}_{\mathbb{M}}$ such that $\mathcal{M}^{\tau}_{\mathbb{M}}(1^{\eta})$ computes in PTIME (a suitable representation of) a sequence $\langle a_1, \ldots, a_n\rangle$ of all elements of $[\![\tau]\!]^{\eta}_{\mathbb{M}}$. When possible, tags are then lifted to arrow types as expected: $\tau_1 \to \tau_2$ is finite (resp. enumerable) when τ_1 and τ_2 are.

Furthermore, well-founded$_{\tau}(<)$ is an additional atom of the logic which requires that the interpretation of the binary function symbol $<$ is deterministic (i.e. $[\![<]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$ does not depends on ρ) and that $([\![\tau]\!]^{\eta}_{\mathbb{M}}, [\![<]\!]^{\eta}_{\mathbb{M},\mathcal{E}})$ is a well-founded set for every η.

D.2 Inference Rules

Inference rules of our proof system, given in Fig. 10, Fig. 11, and Fig. 12, are organized in three categories:

- First, the *structural* rules. This includes weakening rules (of hypotheses, pre- and post-conditions, constraints ...), re-ordering of the terms, and rewriting.
- Second, the *computational* rules. They capture the computations that do not require random samplings or oracle calls from the simulator. This comprises function applications, transitivity, computation of adversarial terms, conditional if then else , computing a function's graphs, and induction.
- Finally, *adversarial* rules capture adversarial capabilities: random samplings and oracle calls.

In order to justify the soundness of our rules, it is useful to have a few lemmas on how validity and/or well-formedness of constraints systems propagate, presents here

D.2.1 Preliminary lemmas for constraints systems. Immedialtly, we can first notice that, for all $\mathbb{M}$ and η and constraints system C^1 and C^2, we have:

- $\mathrm{Valid}(C^1 \cdot C^2) \models \mathrm{Valid}(C^1) \wedge \mathrm{Valid}(C^2)$,
- for all $j \in \{1, 2\}$, $\mathcal{R}^{\eta}_{C^1\cdot C^2,\mathbb{M}} \subseteq \mathcal{R}^{\eta}_{C^j,\mathbb{M}}$, and
- for all $j \in \{1, 2\}$ and random tape $\rho : \mathcal{N}^{\eta,\rho}_{C^j,\mathbb{M}} \subseteq \mathcal{N}^{\eta,\rho}_{C^1\cdot C^2,\mathbb{M}}$

We have similar properties for constraint generalization:

- $\mathrm{Valid}(\prod(x : \tau).\, C) \models \tilde{\forall}(x : \tau).\, \mathrm{Valid}(C)$
- for all $a \in [\![\tau]\!]^{\eta}_{\mathbb{M}}$, $\mathcal{R}^{\eta}_{\prod(x:\tau).C,\mathbb{M}} \subseteq \mathcal{R}^{\eta}_{C,\mathbb{M}[x\mapsto a]}$
- for all $a \in [\![\tau]\!]^{\eta}_{\mathbb{M}}$ and random tape ρ:

$$\mathcal{N}^{\eta,\rho}_{C,\mathbb{M}[x\mapsto a]} \subseteq \mathcal{N}^{\eta,\rho}_{\prod(x:\tau).C,\mathbb{M}}$$

We have similar results for well-formedness properties, for which we provide proofs. In order to prove such properties, we need to

$$\textsc{Weak.Constr}\ \frac{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd \vec{w}_\# \qquad \Theta \models C_\# \subseteq C'_\# \qquad \mathcal{E},\Theta \models_{\mathsf{WF}(\vec{u}_\#)} C'_\#}{\mathcal{E},\Theta,C'_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd \vec{w}_\#}$$

$$\textsc{Weak.Cond}\ \frac{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd f_\#,(v_\# \mid f'_\#),\vec{w}_\# \qquad \mathcal{E},\Theta\vdash [f_\# \Rightarrow f'_\#]_{\mathrm{e}}}{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd (v_\# \mid f_\#),\vec{w}_\#}$$

$$\textsc{Weak.Mem}\ \frac{\mathcal{E},\Theta,C_\#,(\varphi',\psi')\vdash \vec{u}_\# \rhd \vec{v}_\# \qquad \mathcal{E},\Theta\models^A \varphi \Rightarrow \varphi' \qquad \mathcal{E},\Theta\models^A \psi' \Rightarrow \psi}{\mathcal{E},\Theta,C_\#,(\varphi,\psi)\vdash \vec{u}_\# \rhd \vec{v}_\#}$$

$$\textsc{Weak.Hyps}\ \frac{\mathcal{E},\Theta',C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd \vec{w}_\# \qquad \Theta\models\Theta'}{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd \vec{w}_\#}$$

$$\textsc{Definition}\ \frac{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd t_\# \qquad (x:\tau = t_\#)\in\mathcal{E}}{(\mathcal{E}),\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd x}$$

$$\textsc{Refl}\ \frac{}{\mathcal{E},\Theta,\emptyset,(\varphi_\#,\varphi_\#)\vdash \vec{u}_\#,t_\# \rhd t_\#}$$

$$\textsc{Permute}\ \frac{\sigma,\sigma' \text{ are permutations} \qquad \mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash u^1_\#,\cdots,u^n_\# \rhd v^1_\#,\ldots,v^n_\#}{\mathcal{E},\Theta,C_\#,(\varphi,\psi)\vdash u^{\sigma(1)}_\#,\ldots,u^{\sigma(m)}_\# \rhd v^{\sigma'(1)}_\#,\ldots,v^{\sigma'(n)}_\#}$$

$$\textsc{Drop}\ \frac{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd \vec{v}_\#,\vec{t}_\#}{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd \vec{v}_\#}$$

$$\textsc{Rewrite-L}\ \frac{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \#(\vec{w}_0;\vec{w}_1) \rhd v_\# \qquad \mathcal{E},\Theta\vdash [\vec{u}_0=\vec{w}_0]_{\mathrm{e}} \barwedge [\vec{u}_1=\vec{w}_1]_{\mathrm{e}}}{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \#(\vec{u}_0;\vec{u}_1) \rhd v_\#}$$

$$\textsc{Rewrite-R}\ \frac{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd \#(\vec{w}_0;\vec{w}_1) \qquad \mathcal{E},\Theta\vdash [\vec{v}_0=\vec{w}_1]_{\mathrm{e}} \barwedge [\vec{v}_0=\vec{w}_1]_{\mathrm{e}}}{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd \#(\vec{v}_0;\vec{v}_1)}$$

$$\textsc{Dup}\ \frac{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd \vec{v}_\#,\vec{t}_\#}{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd \vec{v}_\#,\vec{t}_\#,\vec{t}_\#}$$

Figure 10: Structural bi-deduction rules

$$\textsc{Name}\ \frac{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd (t_\# \mid f_\#)}{\mathcal{E},\Theta,C_\# \cdot \{(\emptyset,n,t_\#,\mathsf{T}_S,f_\#)\},(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd (n\ t_\# \mid f_\#)}$$

$$\textsc{Oracle}_f\ \frac{\mathcal{E},\Theta,C_\#,(\varphi_\#,\psi_\#)\vdash \vec{u}_\# \rhd_{\mathrm{G}} \vec{w}_\#,(\vec{t}_\# \mid F_\#),(\vec{o}_\# \mid F_\#),(\vec{s}_\# \mid F_\#) \qquad \Theta\models \{\psi_\# \wedge F_\#\} v_\# \leftarrow O_f(\vec{t}_\#)[\vec{k}_\#;\vec{r}_\#]\{\theta_\#\}}{\mathcal{E},\Theta,C'_\#,(\varphi_\#,\theta_\#)\vdash \vec{u}_\# \rhd_{\mathrm{G}} \vec{w}_\#,(v_\# \mid F_\#)}$$

$$\text{with } C'_\# = C_\# \cdot \prod_{v\in f.\mathrm{glob}_\$} (\emptyset,\mathsf{k}_v,o_{v\sharp},\mathsf{T}^{\mathrm{glob}}_{\mathrm{G},v},F_\#) \cdot \prod_{v\in f.\mathrm{loc}_\$} (\emptyset,\mathsf{r}_v,s_{v\sharp},\mathsf{T}^{\mathrm{loc}}_{\mathrm{G}},F_\#);$$
$$\vec{o}_\# = (o_{v\sharp})_{v\in f.\mathrm{glob}_\$} \text{ and } \vec{s}_\# = (s_{v\sharp})_{v\in f.\mathrm{glob}_\$}$$

Figure 11: Adversarial bi-deduction rules

be able to compose the inherent functions coming from the well-formedness definition of different constraints systems, and, as such, we need functions to get the arguments for them. Especially, we need functions to restrict random tapes, which whose existence is guaranteed by the following lemmas (Lemma 3 and Lemma 4).

For the rest of this subsection, we fixed an arbitrary environment $\mathcal{E}$ and an arbitrary formula Θ. All mention of models $\mathbb{M}$ will implicitly be with respect to $\mathcal{E}$, such that $\mathbb{M}:\mathcal{E}\models\Theta$.

Lemma 3. *Let l be a sequence of constraint instances well-formed w.r.t. a model $\mathbb{M}$, $\eta\in\mathbb{N}$ relatively to terms $\vec{u}$. Assume that:*

$$l = ((\vec{a}_0,c_0),\ldots,(\vec{a}_K,c_K))$$

where $c_i = (\vec{\alpha}_i,\mathsf{n}_i,t_i,T_i,f_i))$ for any $i\le K$.

Then, for any $k\in\{0,\ldots,K\}$, there exists a function g_k such that:

$$g_k(\rho_{|\mathbb{M},\eta,l},[\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = [\![(t_k \mid f_k)]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}.$$

Proof. We prove this lemma by induction over k.

Base case (k=0): By well-formedness of l, there exists g such that

$$g(\rho_{|\mathbb{M},\eta,\epsilon},[\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = [\![(t_0 \mid f_0)]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}.$$

where ϵ is the empty sequence. Thus, $\rho_{|\mathbb{M},\eta,\epsilon}$ is the tape where all offsets are set to zero. We conclude by having g_0 be the function that ignores its input tape $\rho_{|\mathbb{M},\eta,l}$ and directly calls g on a zeroed tape and $[\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$.

Inductive case: By well-formedness of l, there exists a function g such that

$$g(\rho_{|\mathbb{M},\eta,l^k},[\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = [\![(t_k \mid f_k)]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}.$$

where $l^k = ((\vec{a}_0,c_0),\ldots,(\vec{a}_k,c_k))$. Furthermore, by induction, for any $i\in\{0,\ldots,k-1\}$ there exists a function g_i such that:

$$g_i(\rho_{|\mathbb{M},\eta,l},[\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = [\![(t_i \mid f_i)]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}.$$

Then there exists then a function g' that, given $\rho_{|\mathbb{M},\eta,l}$ and $[\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$, determines $\rho_{|\mathbb{M},\eta,l^k}$. Indeed, it is possible to determines the set $\mathcal{N}^{\eta,\rho}_{\mathbb{M},l^k}$ using the function $(g_i)_{i<k}$, and then $\rho_{|\mathbb{M},\eta,l^k}$ is determined by zeroing in $\rho_{|\mathbb{M},\eta,l}$ all bit corresponding to name not in $\mathcal{N}^{\eta,\rho}_{\mathbb{M},l^k}$, which is a subset of $\mathcal{N}^{\eta,\rho}_{\mathbb{M},l}$.

We conclude the proof by composing g' with g. □

$$\textsc{Adversarial}\ \frac{\mathcal{E}, \Theta \vdash \mathsf{adv}(t)}{\mathcal{E}, \Theta, \emptyset, (\varphi_\#, \varphi_\#) \vdash \vec{u}_\# \rhd t} \qquad \textsc{FA}\ \frac{\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd \vec{v}_\#, (t_\#^1 \mid f_\#), \ldots, (t_\#^n \mid f_\#) \quad \mathcal{E}, \Theta \vdash \mathsf{adv}(g)}{\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd \vec{v}_\#, (g\ t_\#^1\ \ldots\ t_\#^n \mid f_\#)}$$

$$\textsc{If-then-else}\ \frac{\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd \vec{v}_\#, (b_\# \mid f_\#), (t_\# \mid f_\# \wedge b_\#), (t'_\# \mid f_\# \wedge \neg b_\#)}{\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd \vec{v}_\#, (\mathsf{if}\ b_\#\ \mathsf{then}\ t_\#\ \mathsf{else}\ t'_\# \mid f_\#)} \qquad \textsc{Transitivity}\ \frac{\mathcal{E}, \Theta, C_\#^1, (\varphi_\#, \varphi'_\#) \vdash \vec{u}_\# \rhd \vec{t}_\# \quad \mathcal{E}, \Theta, C_\#^2, (\varphi'_\#, \psi_\#) \vdash \vec{u}_\#, \vec{t}_\# \rhd \vec{v}_\#}{\mathcal{E}, \Theta, C_\#^1 \cdot C_\#^2, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd \vec{t}_\#, \vec{v}_\#}$$

$$\textsc{Lambda-App}\ \frac{\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd (g_\#, t_\# \mid f_\#) \quad \mathcal{E} \vdash \vec{t}_\# : \tau \qquad \mathsf{enum}(\tau)}{\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd (g_\#\ t_\# \mid f_\#)} \qquad \textsc{Lambda}\ \frac{(\mathcal{E}, x : \tau), \Theta, C_\#, (\varphi_\#, \varphi_\#) \vdash \vec{u}_\#, x \rhd (t_\# \mid f_\#) \quad \mathcal{E}, x : \tau \vdash t_\# : \tau_\mathsf{b} \qquad \tau_\mathsf{b} \in \mathbb{B} \qquad \mathsf{enum}(\tau)}{\mathcal{E}, \Theta, \prod_{(x:\tau)} . C_\#, (\varphi_\#, \varphi_\#) \vdash \vec{u}_\# \rhd (\lambda(x : \tau).t_\# \mid f_\#)}$$

$$\textsc{Quantificator-}O \in \{\forall, \exists\}\ \frac{(\mathcal{E}, x : \tau), \Theta, C_\#, (\varphi_\#, \varphi_\#) \vdash \vec{u}_\#, x \rhd (t_\# \mid f_\#) \qquad \mathsf{enum}(\tau)}{\mathcal{E}, \Theta, \prod_{(x:\tau)} C_\#, (\varphi_\#, \varphi_\#) \vdash \vec{u}_\# \rhd (O(x : \tau).t_\# \mid f_\#)}$$

$$\textsc{Induction}\ \frac{(\mathcal{E}, x : \tau), \Theta, C_\#, (\varphi_\#, \varphi_\#) \vdash \vec{u}_\#, (\lambda(y : \tau).\mathsf{if}\ y < x\ \mathsf{then}\ t[x \mapsto y] \mid f_\#), x \rhd (t_\# \mid f_\#) \quad \mathcal{E}, x : \tau \vdash t_\# : \tau_\mathsf{b} \quad \tau_\mathsf{b} \in \mathbb{B} \quad \mathsf{finite}(\tau) \quad \mathsf{fixed}(\tau) \quad \mathcal{E}, \Theta \vdash \mathsf{well\text{-}founded}_\tau(<) \barwedge \mathsf{adv}(<)}{\mathcal{E}, \Theta, \prod_{(x:\tau)} C_\#, (\varphi_\#, \varphi_\#) \vdash \vec{u}_\# \rhd (\lambda(x : \tau).t_\# \mid f_\#)}$$

Figure 12: Computational bi-deduction rules

Finally, the following lemma enable to re-restrict random tape, which is the key lemma for the two main lemmas of this sub-section: Lemma 5 and 6.

Lemma 4. *For any model $\mathbb{M}$ and security parameter η, for any vector of terms $\vec{u}$ for any well-formed sequences of constraint instances l and l' such that all element of l' are in l, there exists a function g such that, for all random tape ρ,*

$$g(\rho_{|\mathbb{M},\eta,l}, [\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = \rho_{|\mathbb{M},\eta,l'}.$$

Proof. Using Lemma 3, we know there exists a function that determines $\mathcal{N}^{\eta,\rho}_{\mathbb{M},l'}$ from $\rho_{|\mathbb{M},\eta,l}$ and $[\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$. Then, the function that zeroes all bits associated to names not in $\mathcal{N}^{\eta,\rho}_{\mathbb{M},l'}$ ends the proof. □

Now, let introduce the lemmas to compose constraints system.

Lemma 5. *For any term $\vec{u}$ and $\vec{v}$, for all constraints system C^1 and C^2 such that*

$$\mathcal{E}, \Theta \models_{WF(\vec{u})} C^1 \quad \text{and} \quad \mathcal{E}, \Theta \models_{WF(\vec{u},\vec{v})} C^2$$

then whenever there exists a function s such that for all logical random tape ρ, and l_{C^1} witnessing the well-formedness of C^1,

$$s(\rho_{|\mathbb{M},\eta,l_{C^1}}, [\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = [\![\vec{v}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$$

then

$$\mathcal{E}, \Theta \models_{WF(\vec{u})} C^1 \cdot C^2.$$

Proof. Let $\mathbb{M}$ be a model w.r.t. $\mathcal{E}$ such that $\mathcal{E}, \Theta \models \mathbb{M}$ and η be a security parameter.

By hypothesis, C^1 is well-formed w.r.t. $\mathbb{M}, \eta$ relatively to $\vec{u}$. Let l_{C^1} be the sequence of constraint instances witnessing its well-formedness relatively to $\vec{u}$. Similarly, let l_{C^2} be the sequence of constraint instances witnessing the well-formedness of C^2 relatively to $\vec{u}, \vec{v}$.

Then, let us show that the concatenation of the two sequences $l_{C^1} \cdot l_{C^2}$ witnesses the well-formedness of $C^1 \cdot C^2$. Let

$$l_{C^1} \cdot l_{C^2} = \big((\vec{a_0}, c_0), \ldots, (\vec{a_K}, c_K)\big)$$

and $(\vec{a_k}, c_k) = (\vec{a_k}, \vec{\alpha}, n, t, T, f)$ be an element of $l_{C^1} \cdot l_{C^2}$.

We must show that:

$$\exists g, \forall \rho, g(\rho_{|\mathbb{M},\eta,l^k}, [\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = [\![(t \mid f)]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha} \mapsto \mathbb{1}^{\eta}_{\vec{a_k}}]:\mathcal{E}} \tag{14}$$

with $l^k = ((\vec{a_1}, c_1), \ldots, (\vec{a_{k-1}}, c_{k-1}))$.

If $(\vec{a_k}, c_k)$ is an element of l_{C^1}, then this is immediate by well-formedness of C^1. Thus, assume that $(\vec{a_k}, c_k)$ is an element of l_{C^2}.

Let then K^1 be the length of l_{C^1}, we have then

$$l_{C^1} = \big((\vec{a_1}, c_1), \ldots, (\vec{a_{K^1}}, c_{K^1})\big)$$

Let ρ be an arbitrary random tape.

By well-formedness of C^2, we have that there exists g_c such that:

$$g_c(\rho_{|\mathbb{M},\eta,l^k_{C^2}}, [\![\vec{u}, \vec{v}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = [\![(t \mid f)]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha} \mapsto \mathbb{1}^{\eta}_{\vec{a_k}}]:\mathcal{E}} \tag{15}$$

with $l^k_{C^2} = [(\vec{a_{K^1+1}}, c_{K^1+1}) \ldots (\vec{a_{k-1}}, c_{k-1})]$

The function g_c is then almost the target function g of Eq. (14) we want. We are left to show there exists functions that return the argument for g_c, and composing these function with g_c will end the proof. Hence, are left to show that

- there exists a function that return $\rho_{|\mathbb{M},\eta,l^k_{C^2}}$ when given $\rho_{|\mathbb{M},\eta,l^k}, [\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$;
- and, for any random tape ρ, there exists a function that return $[\![\vec{v}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$ when given g's argument. However, we have the function s such that $s(\rho_{|\mathbb{M},\eta,l_{C^1}}, [\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = [\![\vec{v}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$, then, it is sufficient to show there exists a function that return $\rho_{|\mathbb{M},\eta,l_{C^1}}$ when given $\rho_{|\mathbb{M},\eta,l^k}, [\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$.

These two points are consequences of Lemma 4. □

Lemma 6. *For all $\mathbb{M} : \mathcal{E}$ and η, for any term $\vec{u}$, any fresh variable x of type τ tagged* enum*, for all constraints system C,*

$$\text{if} \quad \mathcal{E}, (x : \tau), \Theta \models_{WF(\vec{u},x)} C \quad \text{then} \quad \mathcal{E}, \Theta \models_{WF(\vec{u})} \prod_{(x:\tau)} C.$$

Proof. Let $\mathbb{M} : \mathcal{E}$ be a model, and η be a security parameter. For all v in $[\![\tau]\!]^{\eta}_{\mathbb{M}}$, let l^v_C be the sequence witnessing the well-formedness of C w.r.t. $\mathbb{M}_v \stackrel{\text{def}}{=} \mathbb{M}[x \mapsto \mathbb{1}^{\eta}_v]$. We write l^v the sequence obtained from l^v_C by replacing each constraint instance

$$\big(\vec{a}, (\vec{\alpha}, \mathsf{n}, t, T, f)\big) \quad \text{by} \quad \big((v, \vec{a}), ((x, \vec{\alpha}), \mathsf{n}, t, T, f)\big).$$

The type τ is enumerable. Then let $(v_1, \ldots, v_J)$ be a sequence of all elements of $[\![\tau]\!]^{\eta}_{\mathbb{M}}$. Finally, let the concatenation of sequences $l = l^{v_1} \cdots l^{v_J}$ and show that l witnesses the well-formedness of $\prod_{(x:\tau)} C$.

Let j be an integer in $\{1 \ldots J\}$, let then $l^{v_j}_C = ((\vec{a_1}, c_1), \ldots, (\vec{a_K}, c_K))$, and similarly $l^{v_j} = ((v_j, \vec{a_1}, c^x_1), \ldots, (v_j, \vec{a_K}, c^x_K))$, and let k be an integer in $\{1, \ldots K\}$. Let show that there exists a function g, such that for all random tape ρ,

$$g(\rho_{|\mathbb{M},\eta,l^{j,k}}, [\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M},\mathcal{E}}) = [\![(t \mid f)]\!]^{\eta,\rho}_{\mathbb{M}[x \mapsto \mathbb{1}^{\eta}_{v_j}; \vec{\alpha_k} \mapsto \mathbb{1}^{\eta}_{\vec{a}}]}.$$

with $l^{j,k} = l^{v_1} \cdots l^{v_{j-1}} \cdot l^{v_j,k}$ and

$$l^{v_j,k} = \big((v_j, \vec{a_1}, c^x_1), \ldots, (v_j, \vec{a_{k-1}}, c^x_{k-1})\big).$$

By well-formedness of C relatively to $\vec{u}, x$, w.r.t. $\mathbb{M}_{v_j}, \eta$, there exists g_v such that for all random tape ρ,

$$g_v(\rho_{|\mathbb{M}_{v_j},\eta,l^{v_j,k}_C}, [\![\vec{u}, x]\!]^{\eta,\rho}_{\mathbb{M}_{v_j}}) = [\![(t \mid f)]\!]^{\eta,\rho}_{\mathbb{M}_{v_j}[\vec{\alpha_k} \mapsto \mathbb{1}^{\eta}_{\vec{a_k}}]}.$$

with $l^{v_j,k}_C = \big((\vec{a_1}, c_1), \ldots, (\vec{a_{k-1}}, c_{k-1})\big)$ Notice that $\mathbb{M}_{v_j}[\vec{\alpha_k} \mapsto \mathbb{1}^{\eta}_{\vec{a_k}}] = \mathbb{M}[x \mapsto \mathbb{1}^{\eta}_{v_j}; \vec{\alpha_k} \mapsto \mathbb{1}^{\eta}_{\vec{a_k}}]$, and $[\![\vec{u}, x]\!]^{\eta,\rho}_{\mathbb{M}_v} = [\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M},\mathcal{E}}, v$ by term semantic and freshness of x in $\vec{u}$. Then, there is left that there exists a function that for all random tape ρ outputs $\rho_{|\mathbb{M}_{v_j},\eta,l^{v_j,k}_C}$ on inputs $\rho_{|\mathbb{M},\eta,l^{j,k}}, [\![\vec{u}]\!]^{\eta,\rho}_{\mathbb{M},\mathcal{E}}, v_j$ and $\vec{a_k}$. First, notice that in $l^{v_j,k}$, all instances bind the variable x to v_j, then $\mathcal{N}^{\eta,\rho}_{\mathbb{M}_{v_j},l^{v_j,k}_C} = \mathcal{N}^{\eta,\rho}_{\mathbb{M},l^{v_j,k}}$ and thus $\rho_{|\mathbb{M}_{v_j},\eta,l^{v_j,k}_C} = \rho_{|\mathbb{M},\eta,l^{j,k}}$ and we conclude using Lemma 4. □

D.2.2 Structural Rules. The common point of all structural rules is that they do not change the simulator obtained from the premise.

First, notice that a simulator that computes a term $t_\#$, also compute any term $t'_\#$ exactly equal to $t'_\#$. This also holds for input terms: using a term $u_\#$ or a term $u'_\#$ exactly equal does not change the simulator's result. This is captured by rules Rewrite-L and Rewrite-R.

Rule Drop holds because, given a simulator corresponding to the premise, we obtain a simulator for the conclusion by executing the premise simulation and then dropping some of its outputs. Similarly, Permute corresponds to re-ordering inputs and outputs, Refl to copying an input, Dup to duplicating an output, and Definition replaces a variable by its definition.

Then, we have four weakening rules. The rule Weak.Hyps and Weak.Mem for hypothesis and pre- and post-conditions weakening, designed as expected. The rule Weak.Constr for constraints weakening on the same ideas, based on the previous remark that when $C \subseteq C'$ then $\text{Valid}(C') \Rightarrow \text{Valid}(C)$. Finally, the rule Weak.Cond weaken the local formula attached to term. For any term $(t_\# \mid f_\#)$ and a formula $f'_\#$ such that $[f' \Rightarrow f_\#]_e$ then an adversary that compute $(t_\# \mid f_\#)$ and $f'_\#$ can also compute $(t_\# \mid f_\#)$: intuitively, such adversary compute $t_\#$ more "often" than when $f_\#$ is true and at least every time that $f_\#$ is true.

D.2.3 Computational Rules. We call computational rules the rules that do not require random samplings or oracle calls from the simulator:

- the rule Adversarial to build a program that compute an adversarial term (which is immediately an adversary);
- the rules to compute functions (FA for adversarial function, Lambda-App for computed function, If-then-else for specific handling of if then else);
- the rules that chain programs: either with sequence of two programs (Transitivity) or under while loops (Lambda to compute a function graph, Quantificator-$O \in \{\forall, \exists\}$ for specific handling of quantifiers, Induction to compute recursively a function graph). For the proof of these rules one have to show that the final program is still a PTIME adversary, i.e. that the final program is polynomial, correctly chains pre- and post-conditions, and doesn't violate freshness of local samplings and uniqueness of global samplings. For the first point, one need to add restriction on the size of the while loops (we restrict types of lambda terms and quantified variable to be enumerable, or fixed and finite for induction). The second point is ensured by forcing the rules using while loops to apply only on fixed-point conditions. The last point is done by operation on constraints define earlier.

D.2.4 Adversarial Rules. Finally, there are two adversarial rules, which captures two specific capabilities of adversaries: Name corresponds to random samplings; and, Oracle$_f$ to oracle calls.

We give here the definition of an oracle triple validity, which we omitted from the body.

Definition 16 (Oracle Hoare triple)**.** *Consider a oracle triple for an oracle f:*

$$\{\varphi_\#\} v_\# \leftarrow O_f(\vec{t_\#})[\vec{k_\#}; \vec{r_\#}]\{\psi_\#\}$$

whose offsets are of the form

$$\vec{k_\#} = (k_v \; o_{v\sharp})_{v \in f.\text{glob}_\$} \qquad \text{and} \qquad \vec{r_\#} = (r_v \; s_{v\sharp})_{v \in f.\text{loc}_\$}.$$

This triple is valid, i.e.:

$$\Theta \models \{\varphi_\#\} v_\# \leftarrow O_f(\vec{t_\#})[\vec{k_\#}; \vec{s_\#}]\{\psi_\#\},$$

when, for any $\mathbb{M}$ *such that* $\mathbb{M} \models \Theta$, *for any* η, ρ, $\mu_\#$, $i \in \{0,1\}$, *and any fresh variable* X, *if* $\mathbb{M}, \eta, \rho, \mu_i \models^A \varphi_i$ *then* $\mathbb{M}, \eta, \rho, \mu_i' \models^A \psi_i$ *and:*

$$[\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = \mu_i'(X)[f.\text{expr}]^{\eta,\mathfrak{p}}_{\mathbb{M}:\mathcal{E},i,\mu_i'}$$
$$\text{where } \mu_i' = (\!|X \leftarrow O_f([\![\vec{t}_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})[\vec{e}_k\,;\vec{e}_s\,]|\!)^{\eta,\mathfrak{p}}_{\mu_i}$$
$$\vec{e}_k = \left(O_{\mathbb{M}:\mathcal{E},\eta}(k_v, [\![o_{v,i}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})\right)_{v\in f.\text{glob}_\$}$$
$$\vec{e}_s = \left(O_{\mathbb{M}:\mathcal{E},\eta}(r_v, [\![s_{v,i}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})\right)_{v\in f.\text{loc}_\$}$$

where $\mathfrak{p}$ *arbitrary s.t.* ρ_a *is a prefix of its adversarial tape.*

We recall and quickly sketch the proof of Proposition 1.

Proposition 1. *Let* G *be a game and* $f \in \mathcal{O}$ *one of its oracles. The following rule is sound w.r.t. the class of models satisfying* G, *using the notations introduced above:*

$$\textsc{Oracle}_f\ \frac{\mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd_G \vec{w}_\#, (\vec{t}_\# \mid F_\#), (\vec{o}_\# \mid F_\#), (\vec{s}_\# \mid F_\#) \quad \Theta \models \{\psi_\# \wedge F_\#\} v_\# \leftarrow O_f(\vec{t}_\#)[\vec{k}_\#;\vec{r}_\#]\{\theta_\#\}}{\mathcal{E}, \Theta, C'_\#, (\varphi_\#, \theta_\#) \vdash \vec{u}_\# \rhd_G \vec{w}_\#, (v_\# \mid F_\#)}$$

$$\text{with } C'_\# = C_\# \cdot \prod_{v\in f.\text{glob}_\$} (\emptyset, \mathrm{k}_v, o_{v\sharp}, \mathsf{T}^{\text{glob}}_{G,v}, F_\#) \cdot \prod_{v\in f.\text{loc}_\$} (\emptyset, \mathrm{r}_v, s_{v\sharp}, \mathsf{T}^{\text{loc}}_{G}, F_\#)$$
$$\vec{o}_\# = (o_{v\sharp})_{v\in f.\text{glob}_\$} \text{ and } \vec{s}_\# = (s_{v\sharp})_{v\in f.\text{glob}_\$}$$

Proof (sketch). From the bi-deduction premise, we get a simulator p that computes inputs and offsets for the oracle call. The final simulator p′ is p followed by an oracle call. The new program is polynomial if p is, furthermore, the validity of C ensures the freshness of local offsets and uniqueness of global offsets. The equality between the result of p′ and the semantics of the output terms follows from the validity of the Hoare triplet. □

D.3 Proof of Theorem 1

We now recall and prove Theorem 1.

Theorem 1. *Let* $\mathcal{E}$ *be an environment,* Θ *a set of global formulas, and* $\varphi_\#$ *be a bi-assertion such that, for all* $\mathbb{M} : \mathcal{E}$ *satisfying* Θ, *for all* $i \in \{0,1\}$, η, ρ, *we have* $\mathbb{M}, \eta, \rho, \mu^{i\ \eta,\rho}_{init\,\mathbb{M}}(\mathrm{G}) \models^A \varphi_i$. *The following rule is sound w.r.t. models where* G *is secure, for any* $C_\#$, $\#(\vec{v}_0;\vec{v}_1)$ *and* $\psi_\#$*:*

$$\textsc{Bi-deduce}\ \frac{\mathcal{E}, \Theta \vdash \text{Valid}(C_\#) \qquad \mathcal{E}, \Theta, C_\#, (\varphi_\#, \psi_\#) \vdash \emptyset \rhd_G \#(\vec{v}_0;\vec{v}_1)}{\mathcal{E}, \Theta \vdash \vec{v}_0 \sim \vec{v}_1}$$

Proof. Assume that the conclusion is not valid: there exists $\mathbb{M} : \mathcal{E}$ satisfying Θ and a PPTM $\mathcal{D}$ that distinguishes $\vec{v}_0$ from $\vec{v}_1$ with a non-negligible advantage. In other words, the following function is non-negligible:

$$\eta \mapsto \left| Pr_{\rho\in\mathbb{T}_{\mathbb{M},\eta}}\left(\mathcal{D}([\![\vec{v}_0]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, 1^\eta, \rho_a) = 1\right) - Pr_{\rho\in\mathbb{T}_{\mathbb{M},\eta}}\left(\mathcal{D}([\![\vec{v}_1]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, 1^\eta, \rho_a) = 1\right) \right| \tag{16}$$

Assume further that the premises are valid. Since $C_\#$ is valid, we know that C_0 and C_1 are both valid. By the second premise, and by hypothesis on $\varphi_\#$, there exists a PTIME G-adversary p computing $\emptyset \rhd \vec{v}_i$ w.r.t. $\mathbb{M}, \eta, \mathfrak{p}, \rho, \mu_i$ for any $i \in \{0,1\}$, η, ρ and $\mathfrak{p}$ such that $\rho\ \mathcal{R}^\eta_{C_i,\mathbb{M}}\ \mathfrak{p}$, where μ_i is $\mu^{i\ \eta,\rho}_{\text{init}\,\mathbb{M}}$.

We now construct a PTIME G-adversary that wins the game G with non-negligible probability, contradicting the cryptographic assumption. We first translate the distinguisher $\mathcal{D}$ between $\vec{v}_0$ and $\vec{v}_1$ into a program d that performs the same computations[2] for some input variables $\vec{X}$ and return variable res:

$$\text{for all } \eta, i, \mu, \vec{a} \in [\![\vec{\tau}]\!]^\eta_{\mathbb{M}}, \text{ for all tapes } (\rho_a, \rho_h)\ \mathcal{R}^\eta_{C_i,\mathbb{M}}\ \mathfrak{p},$$
$$(\!|\mathrm{d}|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu[\vec{X}\mapsto\vec{a}]}[\text{res}] = \mathcal{D}(\vec{a}, 1^\eta, \rho_a) \tag{17}$$

where $\vec{\tau}$ are the types of $\vec{v}_0$ (and of $\vec{v}_1$, since $\vec{v}_0$ and $\vec{v}_1$ have the same types). Since $\mathcal{D}$ only accesses ρ_a, the program d only accesses $\mathfrak{p}[\mathsf{T}_A, \text{bool}]$, hence it is an adversary. We can thus form an adversary q by composing d with p. C_0 is well-formed, hence, by Lemma 1, there exists a coupling $\mathbb{C} : \mathbb{T}_{\mathbb{M},\eta} \bowtie \mathfrak{P}$ contained in $\mathcal{R}^\eta_{C_i,\mathbb{M}}$.

Hence using Eq. (17) and Lemma 2, we obtain that:

$$Pr_{\mathfrak{p}}\left((\!|\mathrm{q}|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},0,\mu_0}[\text{res}] = 1\right) = Pr_{\rho\in\mathbb{T}_{\mathbb{M},\eta}}\left(\mathcal{D}([\![\vec{v}_0]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, 1^\eta, \rho_a) = 1\right).$$

Repeating the reasoning for $i = 1$, we obtain that the following function is equal to the one in Eq. (16):

$$\eta \mapsto \left| Pr_{\mathfrak{p}}\left((\!|\mathrm{q}|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},0,\mu_0}[\text{res}] = 1\right) - Pr_{\mathfrak{p}}\left((\!|\mathrm{q}|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},1,\mu_1}[\text{res}] = 1\right) \right|.$$

It is thus non-negligible, contradicting the assumption on G in $\mathbb{M}$. □

E IMPLEMENTATION

We detail here the fully automated procedure $\text{search}_\rhd$ for finding bi-deduction proofs using the proof system of Section 4. As explained in section Section 5, we aim at a fully automated procedure, reasonably efficient, but not necessarily complete. Then, rather than requiring user guidance during proof search, our algorithm will make heuristic choices on how to build the proof, and return to the user some proof obligations (as global and local formulas) that can then be proved, automatically or not, using the standard means in the proof assistant.

E.1 Recursive Functions

Our logic supports recursively defined functions, which are crucially used to model protocols, as shown in Section 2. Following this particular use case, we will consider only recursive functions over timestamps, even though our approach is more general than that. If $f_\#$ is recursively defined, proving that a term $(f_\#\ u_\#)$ is bi-deducible will often require to prove that $(f_\#\ x_\#)$ can be bi-deduced for all values $x_\# \le u_\#$. This can only be achieved in our proof-system using the induction rule, which is notoriously difficult to automate due to the need to find generalizations that are invariant. In our case, invariants concerns the assertions, but also the terms to be deduced.

Instead of searching for proofs using induction rules in arbitrary ways, our approach is to look for proofs that follow a simple construction pattern. This eases automation and provides good results in practice, see Section 6. When trying to verify a bi-deduction judgement $\vec{u}_\# \rhd \vec{v}_\#$, we first search for induction-free derivations of judgements of the following form (we will discuss later how these

[2] In this equation, i and μ are arbitrary, because our program initializes its own memory and does not call any oracle.

judgments are determined):

$$\begin{array}{l}(\mathcal{E}, t : \mathsf{timestamp}), \Theta, C^i_\#, (\varphi_\#, \varphi_\#) \vdash \\ \quad \vec{u}_\#, (\lambda t'.\ \text{if } t' < t \wedge t \le t_0 \text{ then } \vec{w}_\#[t \mapsto t']), t \rhd (w^i_\# \mid t \le t_0) \\ \qquad \mathcal{E}, \Theta, C'_\#, (\varphi_\#, \psi_\#) \vdash \vec{u}_\#, (\lambda t.\ \text{if } t \le t_0 \text{ then } \vec{w}_\#) \rhd \vec{v}_\#\end{array} \tag{18}$$

The rough idea is that, instead of bi-deducing $\vec{u}_\# \rhd \vec{v}_\#$, we more generally try to bi-deduce:

$$\vec{u}_\# \ \rhd (\lambda t.\ \text{if } t \le t_0 \text{ then } \vec{w}_\#), \vec{v}_\#$$

for some well-chosen $\vec{w}_\# = (w^1_\#, \ldots, w^k_\#)$. By transitivity, the extra terms are first bi-deduced by induction, and then become available to ease the bi-deduction of $\vec{v}_\#$. More precisely, we derive:

$$\mathcal{E}, \Theta, C'_\# \cdot \forall t.(\textstyle\prod_{i\in[1;k]} C^i_\#), (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \rhd \vec{v}_\#$$

using rules Transitivity, Induction and Drop and the derivations of the above judgements, with a derivation of Eq. (18) for each $i \in [1;k]$. Note that we restrict to have the same pre- and post-condition φ on all judgments of Eq. (18). More generally, our invariants are insensitive to the iteration variable t, which can be limiting, but generally keeps our proof search procedure reasonably simple and notably helps termination.

In our implementation, $\vec{w}_\#$ is obtained from $\vec{v}_\#$ by collecting all recursive definitions that may be useful, directly or indirectly, to deduce the recursive definitions appearing in $\vec{v}_\#$. This is done using a fixed-point computation, introduced in [6], which balances efficiency and precision.

Example 10. *In Section 2 we seek to prove the following indistinguishability, where* $\mathsf{n}_{\mathsf{fresh}}$ *is a name which is not used in the protocol:*

$$\begin{array}{l}\Theta \models \ \mathsf{frame}(\mathsf{pred\ t_0}), \mathsf{n}\ (i_0, j_0), \mathsf{h}(\langle \mathsf{n}\ (i_0, j_0), \mathsf{att}(\mathsf{frame}(\mathsf{pred\ t_0}))\rangle, \mathsf{k}\ i_0) \\ \quad \sim \ \mathsf{frame}(\mathsf{pred\ t_0}), \mathsf{n}\ (i_0, j_0), \mathsf{n}_{\mathsf{fresh}}\end{array}$$

with $t_0 = \mathsf{T}(i_0, j_0)$*. Also, the indices* i_0 *and* j_0 *are assumed adversarial in* Θ*. Our indistinguishability actually follows from the following bi-deduction judgement w.r.t. the PRF game, for any constraint system* C *that is valid and any pre-condition* φ *that holds on the game's initial memory:*

$$\begin{array}{l}\mathcal{E}, \Theta, C, (\varphi, \psi) \vdash \emptyset \rhd \\ \quad \mathsf{frame}(\mathsf{pred\ t_0}), \mathsf{n}\ (i_0, j_0), \\ \quad \text{if } f_{\mathrm{Fresh}} \text{ then} \\ \qquad \#(\mathsf{h}(\langle \mathsf{n}\ (i_0, j_0), \mathsf{att}(\mathsf{frame}(\mathsf{pred\ t_0}))\rangle, \mathsf{k}\ i_0); \mathsf{n}_{\mathsf{fresh}})\end{array}$$

where f_{Fresh} *is the (overwhelmingly true) formula stating that*

$$\mathsf{n}(i_0, j_0) \neq \mathsf{n}(i, j)$$

for all i, j *such that* $\mathsf{T}(i, j) < \mathsf{T}(i_0, j_0)$*. To simplify the presentation, we omit markers indicating that* C*,* φ *and* ψ *are actually bi-constraint systems and bi-assertions.*

We choose $\vec{w}_\# = (\mathsf{frame(t)}, \mathsf{input(t)}, \mathsf{output(t)})$ *to prove this judgment using the strategy defined above, Let:*

$$w'_\# = \big(\lambda t'.\ \text{if } t' < t \text{ then } (\mathsf{frame(t')}, \mathsf{input(t')}, \mathsf{output(t')}) \mid t \le \mathsf{pred}(t_0)\big).$$

We will thus have to prove the following judgments, for some constraint systems C_1, C_2, C_3, C' *and:*

$$\begin{array}{l}(\mathcal{E}, t : \mathsf{timestamp}), \Theta, C_1, (\varphi, \varphi) \vdash w'_\#, t \rhd (\mathsf{frame(t)} \mid t \le \mathsf{pred}(t_0)) \\ (\mathcal{E}, t : \mathsf{timestamp}), \Theta, C_2, (\varphi, \varphi) \vdash w'_\# \rhd (\mathsf{input(t)} \mid t \le \mathsf{pred}(t_0)) \\ (\mathcal{E}, t : \mathsf{timestamp}), \Theta, C_3, (\varphi, \varphi) \vdash w'_\# \rhd (\mathsf{output(t)} \mid t \le \mathsf{pred}(t_0))\end{array}$$

$$\begin{array}{l}\mathcal{E}, \Theta, C', (\varphi, \psi) \vdash (\lambda t.\ \text{if } t \le \mathsf{pred}(t_0) \text{ then } \vec{w}_\#) \rhd \\ \qquad \mathsf{frame}(\mathsf{pred\ t_0}), \mathsf{n}\ (i_0, j_0), \text{if } f_{\mathrm{Fresh}} \text{ then } \#(\mathsf{output}(\mathsf{t_0}); \mathsf{n}_{\mathsf{fresh}})\end{array}$$

This kind of generalization will occur very often when bi-deduction involves inputs, outputs, or frames, due to the mutual definition of these functions. These judgments can indeed be proved, without using the induction rule; details are given in Example 13.

This example leaves unspecified the assertion language, and does not explain how assertions can be synthesized to fit the proof's requirements; this is explained next.

E.2 Assertion Language

Most cryptographic games rely only on global variables to store monotonic logs, which are then used only to check that some messages have not been logged. We choose an assertion language that is well adapted to this use case. This is enough to support all cryptographic games already supported by Squirrel and, arguably, most standard cryptographic games. In term of operations supported in the cryptographic games, this yields a similar expressivity to CryptoVerif, which supports logs through tables.

Given that oracles and simulators are programs, we cannot hope to precisely characterize the game's memory at any point of a bi-deduction. In particular, proofs by induction will require assertions that are invariant by bi-deduction steps. Finding invariant assertions can be achieved by sufficiently over-approximating the logged values in assertions. The over-approximation should just be precise enough to be able to ensure that some values are absent from the logs.

Concretely, we use assertions that associate, to each log of the game, a formal union of elements of the form $(\vec{\alpha}, m_\#, f_\#)$ where $\vec{\alpha}$ is a list of variables, $m_\#$ is a term and $f_\#$ is a local formula. As for constraints, $\vec{\alpha}$ should be understood as a binders. The semantics of an item $(\vec{\alpha}, m, f)$ w.r.t. $\mathbb{M}, \eta, \rho$ is a set of possible semantic values, where $\vec{\tau}$ is the vector of the types of $\vec{\alpha}$:

$$[\![(\vec{\alpha}, m, f)]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} \stackrel{\text{def}}{=} \left\{ [\![m]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha}\mapsto\vec{a}]:\mathcal{E}} \;\middle|\; \vec{a} \in [\![\vec{\tau}]\!]^{\eta}_{\mathbb{M}}, [\![f]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha}\mapsto\vec{a}]:\mathcal{E}} = 1 \right\}$$

The semantics for a formal union of such items is then naturally taken as the union of the semantics of all items. Finally, we define the semantics of an assertion by considering that it is satisfied by a memory when the semantic values in each log are contained in the semantics of the corresponding formal set in the assertion:

$$\mathbb{M}, \eta, \rho, \mu \models^A \varphi \text{ when, for all } \ell \text{ log of G, } \mu(\ell) \subseteq [\![\varphi(\ell)]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$$

This assertion language supports the required operations: given a local formula f and an assertion φ, we define $\varphi \wedge f$ as the assertion where f is added as a conjunction to each condition in φ; we can also define $\varphi \cup \psi$ as the point-wise union of assertions, which over-approximates $\varphi \vee \psi$. Finally, assertions can be generalized over

some variable: we define $\forall x.\varphi$ as the assertion φ where x has been added to the first component of each item.

Example 11. *To obtain a complete derivation in Example 10, we can use the following assertions:*

$$\varphi(\ell_{\mathsf{hash}}) = \{(\{i,j\}, \mathsf{n}\ (i,j), \mathsf{T}(i,j) \le \mathsf{pred}(t_0))\}$$
$$\varphi(\ell_{\mathsf{challenge}}) = \emptyset \qquad \psi(\ell_{\mathsf{hash}}) = \varphi(\ell_{\mathsf{hash}})$$
$$\psi(\ell_{\mathsf{challenge}}) = \{(\emptyset, \mathsf{n}\ (i_0, j_0), \mathsf{true})\}$$

The invariant φ over-approximates the messages passed to the hash oracle during the computation of $\vec{w}$; it could be made precise by taking $(\{j\}, \mathsf{n}, (i_0, j), \mathsf{T}(i_0, j) < t_0)$. It is however precise enough to allow the final use of the oracle rule for the challenge oracle, which requires that $\mathsf{n}\ (i_0, j_0)$ has not previously been used in an oracle query.

E.3 Proof Search

In order to find the induction-free derivations required in the general proof method described in Appendix E.1, we implement the goal-directed heuristic proof search procedure $\mathsf{search}_{\triangleright}(\cdot)$ that takes as input a *partial bi-deduction goal*, with an incomplete *well-formed* constraint system and global hypotheses and without a post-condition, and finds a derivation for a possible completion of this goal. More precisely, we must have that for any initial environment $\mathcal{E}$, hypotheses Θ, input terms $\vec{u}_\#$, constraints $C_\#$ such that $\mathcal{E}, \Theta \models_{\mathsf{WF}(\vec{u}_\#)} C_\#$, pre-condition $\varphi_\#$, and output terms $\vec{v}_\#$,

$$\mathsf{search}_{\triangleright}(\mathcal{E}, \Theta, C_\#, (\varphi_\#, \cdot) \vdash \vec{u}_\# \triangleright (\vec{v}_\# \mid f_\#)) = (\Theta', C'_\#, \psi_\#, f'_\#)$$
implies that
$$\mathcal{E}, \Theta \cup \Theta', (C_\# \cdot C'_\#), (\varphi_\#, \psi_\#) \vdash \vec{u}_\# \triangleright (\vec{v}_\# \mid f_\# \wedge f'_\#) \text{ is derivable}$$
and $$\mathcal{E}, \Theta \models_{\mathsf{WF}(\vec{u}_\#)} C_\# \cdot C'_\#$$

In case the proof search fails, $\mathsf{search}_{\triangleright}(\cdot)$ returns an error. The addition of Θ' and $f'_\#$ corresponds to proof obligations, at different levels, that the user may discharge later on. Our procedure makes use of existing automated deduction capabilities in Squirrel to automatically verify formulas when needed; it is notably used to limit the number of produced proof obligations. In Example 10, the condition f_{Fresh} would be added as $f'_\#$, and could be automatically discharged later.

For usability and performance reasons, our proof search procedure is fully deterministic, guided by the structure of the terms to deduce. It eagerly applies the Refl and Adversarial rules whenever possible. It also eagerly applies the oracle rule when constraints in $C_\#$ allow it. When this is not guaranteed, a specific strategy is used to avoid abusive use of oracles that may prevent the completion of the rest of the proof. We illustrate it with the PRF game, in a situation where we want to deduce a message $\mathsf{h}(m, \mathsf{k}\ i)$, the constraint system specifies that the game's key corresponds to $(\mathsf{k}\ j)$, but we cannot check that $i = j$: in this case we rewrite the term to be deduced into if $i = j$ then $\mathsf{h}(m, \mathsf{k}\ i)$ else $\mathsf{h}(m, \mathsf{k}\ i)$ and use the conditional rule If-then-else, after which we can use the hash oracle to deduce $(\mathsf{h}(m, \mathsf{k}\ i) \mid i = j)$ but directly compute $(\mathsf{h}(m, \mathsf{k}\ j) \mid i \neq j)$ by having the simulator sample $(\mathsf{k}\ j)$,

When applying oracle rules, our strategy needs to synthesize a post-condition from the pre-condition, such that the Hoare triple is valid. Because our assertions only track logs, which are assume to evolve only monotonically, this can be done by enriching (by means of unions) the pre-condition, e.g. taking $\psi(\ell) = \varphi(\ell) \cup (\emptyset, m_\#, f_\#)$ when an oracle is called on $m_\#$ under condition $f_\#$.

E.4 Invariant Synthesis

In order to find the initial pre-condition φ required to complete the whole proof by induction of Appendix E.1, we proceed iteratively, attempting to find an invariant φ as the result of a fixed-point computation. We start with the assertion φ^0 stipulating that all logs are empty. At iteration i, we will have an assertion φ^i, and we use our proof search procedure to complete induction-free derivations. If this succeeds, it yields several post-conditions ψ, which can be regarded as a single post-condition by taking their union. We define φ^{i+1} as $\varphi^i \cup \forall t.\psi$ – this generalization is necessary for the new condition to make sense w.r.t. $\mathcal{E}$ and, intuitively, to express that the post-condition enriches the pre-condition with new potential logged terms for all values of t. Finally, we check whether this new condition semantically entails the previous one. If this is the case, we have found an invariant – and derivations of the expected form, with φ^i as post-conditions, can be obtained by weakening post-conditions to φ^i. If this is not the case, we move on to the next iteration to further over-approximate our condition.

The idea that assertions over-approximate logs, and this fixed-point computation of invariants, is directly inspired by abstract interpretation techniques [25]. In this regard, our assertion language forms a rather crude abstract domain, though it already provides good results. Our algorithms might be improved in the future by using richer abstract domains for sets of terms, e.g. [34, 40].

Example 12. *In the proof of Example 10, starting with φ^0 in which $\varphi^0(\ell_{\mathsf{challenge}}) = \varphi^0(\ell_{\mathsf{hash}}) = \emptyset$, the first derivation of our three inductive bi-deduction judgements yields, after some simplifications,*

$$\varphi^1(\ell_{\mathsf{hash}}) = \big(\{t, i, j\}, \mathsf{n}\ (i,j), t = \mathsf{T}(i,j) \le \mathsf{pred}(t_0)\big)$$

and $\varphi^1(\ell_{\mathsf{challenge}}) = \emptyset$. Since $\not\models \varphi^0 \Rightarrow \varphi^1$, we restart the proof-search for our three judgements with φ^1 as the new pre-condition. We obtain $\varphi^2 = \varphi^1$, at which point we can search for the fourth derivation. This will succeed with a post-condition mentioning the use of the challenge oracle. It will finally remain to check that the produced constraints are valid, to conclude that bi-deduction holds.

Example 13. *We explain how the judgements obtained at the end of Example 10 can be derived. To derive the final judgement, recall that* $\mathsf{output}(\mathsf{t}_0) = \mathsf{h}(\langle \mathsf{n}\ (i_0, j_0), \mathsf{att}(\mathsf{frame}(\mathsf{pred}\ \mathsf{t}_0))\rangle, \mathsf{k}\ i_0)$. *Because* att *is adversarial and* $\mathsf{frame}(\mathsf{pred}\ \mathsf{t}_0)$ *is available from $\vec{w}_\#$ in input, we can derive the first term, as well as the second component of the hashed message. The first component of the hashed message can also be derived by simulator's random sampling, which would add the constraints $(\emptyset, \mathsf{n}, (i_0, j_0), \mathsf{T}_S, \top)$. To conclude, we have to call the hash oracle, which adds the constraint $(\emptyset, k, i_0, \mathsf{T}^{\mathsf{glob}}_{\mathsf{G},\mathsf{key}})$, and requires that φ implies that the hashed message is not in $\ell_{\mathsf{hash}} \cup \ell_{\mathsf{challenge}}$.*

Because the constraint systems C_i will eventually be combined with C' to yield a system that must be valid, we will only be able to use the name $\mathsf{k}\ i_0$ as the game's key in the other bi-deduction derivations.

The input judgement above can easily be derived, taking $C_2 = \emptyset$, by definition of the input function and because the frame at $\mathsf{pred}\ t < t$ *is available from w'. Similarly, we easily obtain the frame judgement, with $C_1 = \emptyset$.*

For the output judgement, we also expand the definition of the output function, and proceed by case analysis over t. In the key case, we have to derive:

$$(\mathcal{E}, t, i, j), \Theta \tilde{\wedge} [t = \mathsf{T}(i,j)]_{\mathrm{e}}, C_3', (\varphi, \varphi) \vdash w' \rhd \mathsf{h}(\langle \mathsf{n}\ (i,j), \mathsf{input}(\mathsf{t})\rangle, \mathsf{k}\ i)$$

Clearly, the hashed message can be bi-deduced with:

$$(\{i,j\}, \mathsf{n}, (i,j), \mathsf{T}_S, \top) \in C_3'.$$

To compute the hash per se, there are two cases:

- *Either $i_0 \neq i$, in which case the adversary can sample the key $k\ i$ and compute the hash itself. For that case, we add $(\{i\}, k\ i, \mathsf{T}_S, i_0 \neq i)$ in the constraints system.*
- *Or $i_0 = i$, in which case the only way for the adversary to compute the hash without rendering the constraints invalid is by calling the hashing oracle. Doing so adds the constraint $(, \mathsf{n}, (i_0, j), \mathsf{T}_S, \top)$, and requires that the condition φ is invariant by the oracle call, i.e. preserved by the addition of the hashed message to ℓ_{hash}.*

Overall, we can derive the output judgement with:

$$C_3 = \{(\{i,j\}, \mathsf{n}, (i,j), \top),\ (\emptyset, k\, i_0, \top)\}.$$