Cryptographic Reductions By Bi-Deduction

David Baelde, Adrien Koutsos, Justine Sauvage

Cryptographic Reductions By Bi-Deduction

David Baelde*, Adrien Koutsos† and Justine Sauvage†
* Univ Rennes, CNRS, IRISA, France
† Inria Paris, France

***Index Terms*—Cryptographic Protocols, Cryptographic Games, Provable Security, Logic.**

***Abstract*—In this paper, we are interested in the Computationally Complete Symbolic Attacker (CCSA) approach to security protocol verification, which relies on probabilistic logics to represent and reason about the interaction traces between a protocol and an arbitrary adversary. The proof assistant SQUIRREL implements one such logic. The 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 in this paper, by providing a formal and systematic method for deriving axioms from cryptographic games. Our method relies on synthesizing (parts of) an adversary w.r.t. some cryptographic game, through the notion of bi-deduction. Concretely, we define a rich notion of bi-deduction, justify how to use it do derive cryptographic axioms, provide a proof system for bi-deduction, and an automatic proof-search method which we implemented in SQUIRREL.**

I. INTRODUCTION

Computer systems are being used for increasingly many applications in our digitalized societies: messaging, payments, access control, voting, etc. All these applications involve communication protocols, themselves relying on cryptographic primitives, which enable the implementation of the desired functionality while ensuring 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 thus been the topic of intense research over the past decades. In this paper, we are concerned with obtaining formal proofs of abstract protocols. 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], [2]). 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 (see, e.g., [4], [5] for striking attacks found at this level of abstraction, and [6] 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 to establish, 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.

Example 1. *The* Pseudo-Random *Family (PRF) assumption requires that a cryptographic hash functions cannot be distinguished from a random function. We present here a variation of the standard PRF game of cryptographers, which yields an equivalent assumption but is easier to use to obtain practical CCSA axioms, as shown later. Consider two cryptographic games* G_0 *and* G_1 *relying on a* security parameter $\eta \in \mathbb{N}$*:*

- *Both games are initialized by sampling uniformly a key* k *of length* η*, and setting* $\ell = []$.
- *In both games, a* hashing oracle *can be queried any number of times. For each input* m*, it outputs* $\mathsf{h}(m, \mathsf{k})$ *after having logged the query by setting* $\ell := m :: \ell$.
- *A* real-or-random challenge oracle *is finally provided, which can only be queried once, at the end of the game, on a message* $m \notin \ell$*. In* G_0*, this oracle will output* $\mathsf{h}(m, \mathsf{k})$*. In* G_1*, it will output a freshly sampled value of the same length.*

The advantage *of a PPTM adversary* $\mathcal{A}$ *in* $\mathsf{G} = (\mathsf{G}_0, \mathsf{G}_1)$ *is the probability that it correctly guesses with which game it is interacting:* $\left|Pr(\mathcal{A}^{\mathsf{G}_0} = 1) - Pr(\mathcal{A}^{\mathsf{G}_1} = 1)\right|$*. The hash function is said to be PRF when this advantage is negligible — i.e. asymptotically smaller than* η^{-k} *for any* $k \in \mathbb{N}$ *— for any PPTM* $\mathcal{A}$

Proofs in the computational model typically proceed by "game hopping" [7]: 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 [8] automates common game-hopping transformations, Owl [9] allows to prove reachability properties by typing, while other tools (Easycrypt [10], CryptHOL [11], SSProve [12]) allow reasoning through (probabilistic and relational) program logics.

[13] 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 [14], [15], [16], [17] and later using the proof assistant SQUIRREL, which relies on enriched CCSA logics that notably internalize the notions of protocol and execution traces. Initially proposed as a CCSA *meta-logic* [18], [19], the logic behind SQUIRREL is now a more general, *higher-order* CCSA logic [20].

Example 2. *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 cryptographers' PRF is expressed through the following axiom scheme [15]:*

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

Here, $\vec{u}$ and v must be closed terms; H is the set of all terms m such that $h(m, \mathsf{k})$ 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 game of Example 1 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 real-or-random 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 corresponding 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$_1$) [20, 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.

a) 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 hypothesis — represented by the game $\mathsf{G} = (\mathsf{G}_0, \mathsf{G}_1)$ — whenever the terms $\vec{u}_0$ and $\vec{u}_1$ can be computed by an attacker 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) 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. [19] introduced the notion of bi-deduction: a judgment $(\vec{v_0}, \vec{v_1}) \rhd (\vec{u_0}, \vec{u_1})$ 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 $(\emptyset, \emptyset) \rhd (\vec{u_0}, \vec{u_1})$ 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 [19] is very limited: first, no interaction (through oracle calls) with an external game are 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 [18], [20] are all completely out-of-scope.

b) 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 [19] 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. E.g., 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 real-or-random oracle, corresponding to the ad-hoc conjunction over H in Example 2.

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 implemented this proof-search algorithm in SQUIRREL, to validate our approach on several case studies.

c) Related Work: The deduction problem has been extensively studied in the literature, albeit in different settings. E.g. [21], [22], [23] 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 [24], 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.

d) Outline: We present in Section II the fragment of SQUIRREL's higher-order CCSA logic in which we shall work, and define in Section III a formal model of cryptographic games and adversaries that is adapted for our needs. Our notion of bi-deduction is then presented in Section IV, together with a proof system for deriving bi-deducibility. Finally, Section V presents a proof-search procedure for this system. We report in Section VI on some case studies performed with an extension of SQUIRREL implementing this procedure (now merged into the main branch of the tool [25]).

II. LOGIC

We recall the main features of the indistinguishability logic of [20]. 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 (hence terms can represent *dependent* random variables). The logic features predicates capturing specific properties of the random variable interpretations of terms. For instance, $t_1 \sim t_2$ holds when the advantage of any PPTM in distinguishing t_1 from t_2 is negligible.

A key aspect is that the probabilistic semantics of terms is represented as deterministic functions taking as input tapes providing the source of randomness. 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. This makes all sources of randomness explicit, and allows to easily track the randomness shared between different computations.

a) Types: We consider simple types over a set of base types $\mathbb{B}$ (e.g. bool, $\mathsf{message}$, 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.

b) Type Structures: A type structure $\mathbb{M}$ provides interpretation domains for types, which are parameterized by the security parameter η. For every $\eta \in \mathbb{N}$, every base type $\tau_\mathsf{b} \in \mathbb{B}$ maps to a set $[\![\tau_\mathsf{b}]\!]^\eta_\mathbb{M}$, which is the interpretation domain of terms of type τ_b for a specific value of η. The interpretation is then lifted to arrow types as expected: $[\![\tau_1 \to \tau_2]\!]^\eta_\mathbb{M} = [\![\tau_1]\!]^\eta_\mathbb{M} \to [\![\tau_2]\!]^\eta_\mathbb{M}$, i.e. the set of functions from $[\![\tau_1]\!]^\eta_\mathbb{M}$ to $[\![\tau_2]\!]^\eta_\mathbb{M}$.

We require that standard types are interpreted as expected, e.g. bool is interpreted as $\{0,1\}$ and int as $\mathbb{N}$. We also require that $\mathbb{M}$ specifies the sampling procedures used to sample values in each type. First, for every base type $\tau_\mathsf{b} \in \mathbb{B}$ and η, $\mathbb{M}$ defines the number $\mathsf{R}_{\mathbb{M},\eta}(\tau_\mathsf{b}) \in \mathbb{N}$ of random bits needed to sample a value of type τ_b. Second, $\mathbb{M}$ provides a machine $[\![\tau_\mathsf{b}]\!]^\$_\mathbb{M}$ such that for every η and bitstring w of length $\mathsf{R}_{\mathbb{M},\eta}(\tau_\mathsf{b})$, $[\![\tau_\mathsf{b}]\!]^\$_\mathbb{M}(1^\eta, w)$ computes a value in $[\![\tau_\mathsf{b}]\!]^\eta_\mathbb{M}$ in time polynomial in η.

c) Terms: We consider simply-typed λ-terms built from a set of variable symbols $\mathcal{X}$:

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

As usual, terms are taken modulo alpha-renaming, and we let $\mathsf{fv}(t)$ denotes the free variables of t. A term is typed and interpreted w.r.t. an *environment* $\mathcal{E}$, which is a sequence of: *declarations* of the form $(x : \tau)$, which state the existence of a variable x of type τ; *definitions* of the form $(x : \tau = t)$, which introduce new variables with a fixed meaning. Recursive definitions are allowed, with a semantic well-foundedness condition [20] whose details are irrelevant for the present work. We write $\mathcal{E} \vdash t : \tau$ when t has type τ in $\mathcal{E}$, and we only consider well-typed terms and environments (see [20] for typing rules).

d) Models: A *term structure*, or *model* $\mathbb{M}$ for an environment $\mathcal{E}$, which we write $\mathbb{M} : \mathcal{E}$, allows to interpret well-typed terms as random variables on the interpretation of their types, as we describe next. The model $\mathbb{M}$ first specifies, for every η, a sample space $\mathbb{T}_{\mathbb{M},\eta} = \mathbb{T}^\mathsf{a}_{\mathbb{M},\eta} \times \mathbb{T}^\mathsf{h}_{\mathbb{M},\eta}$, where $\mathbb{T}^\mathsf{a}_{\mathbb{M},\eta}$ (resp. $\mathbb{T}^\mathsf{h}_{\mathbb{M},\eta}$) is the set of all bit-strings of some length fixed by $\mathbb{M}$. Elements of $\mathbb{T}_{\mathbb{M},\eta}$ are pairs $\rho = (\rho_\mathsf{a}, \rho_\mathsf{h})$ of *random tapes*: 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 the sample space $\mathbb{T}_{\mathbb{M},\eta}$ to $[\![\tau]\!]^\eta_\mathbb{M}$:

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

For each declaration $(x : \tau) \in \mathcal{E}$, the model $\mathbb{M}$ provides an interpretation $\mathbb{M}(x) \in \mathbb{RV}_\mathbb{M}(\tau)$. 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 of declared variables is finally lifted to define $[\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} \in \mathbb{RV}_\mathbb{M}(\tau)$ for any term t such that $\mathcal{E} \vdash t : \tau$, as follows:

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

$$[\![t\ t']\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} \stackrel{\text{def}}{=} [\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}([\![t']\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})$$
$$[\![\lambda(x:\tau).t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} \stackrel{\text{def}}{=} \left(a \in [\![\tau]\!]^{\eta}_{\mathbb{M}} \mapsto [\![t]\!]^{\eta,\rho}_{\mathbb{M}[x\mapsto \mathbb{1}^{\eta}_{a}]:(\mathcal{E},x:\tau)}\right)$$

where $\mathbb{1}^{\eta}_{a}$ is the element of $\mathbb{RV}_{\mathbb{M}}(\tau)$ 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.

If $\mathcal{E}$ contains a definition $x : \tau = t$, the interpretation of x is given by that of t, i.e. we have $[\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = [\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$. The well-foundedness conditions on recursive definitions ensure that this yields a unique interpretation.

e) Global Formulas: The formulas of our first-order logic are as usual. We call them *global formulas* and decorate their logical connectives to distinguish them from another language of formulas that we will introduce later:

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

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 formulas (i.e. the satisfaction relation $\mathbb{M} \models F$) derives as usual from the semantics of atomic formulas (the last four constructs) which we describe next.

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. $\mathcal{M}(1^\eta, \rho_{\mathsf{a}}) = [\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$ for all $\eta \in \mathbb{N}$ and $\rho = (\rho_{\mathsf{a}}, \rho_{\mathsf{h}}) \in \mathbb{T}_{\mathbb{M},\eta}$. The definition is extended naturally to terms t of type $\tau^1_{\mathsf{b}} \to \ldots \to \tau^k_{\mathsf{b}} \to \tau_{\mathsf{b}}$ by requiring the existence of a PPTM taking function arguments as inputs.

For any boolean term t, the atoms $[t]_{\mathrm{e}}$ and $[t]$ state, respectively, that t is exactly true and *overwhelmingly* true:

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

Finally, if $\vec{t} = t_1, \ldots, t_n$ and $\vec{t'} = t'_1, \ldots, t'_n$ are two are sequences of terms with compatible base types (i.e. t_i and t'_i have the same types 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:

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

Following standard notation 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 $\mathcal{E}, \emptyset \models F$. Finally, judgements of the logic are of the form $\mathcal{E}, \Theta \vdash F$ (where Θ, F are well-typed in $\mathcal{E}$). Such a judgement is valid whenever $\tilde{\wedge}\Theta \tilde{\Rightarrow} F$ is valid.

Example 3. *For any $t : \mathsf{bool}$ we have $\mathbb{M} \models [t]_{\mathrm{e}} \tilde{\Rightarrow} [t]$ for any $\mathbb{M}$, i.e. that formula is valid. The converse implication is not valid. Moreover, $[t]$ is logically equivalent to $t \sim \mathsf{true}$.*

f) Builtins and Local Formulas: We will need to restrict to models satisfying some conditions. We first assume that environments always contain some builtins, including true, false, if $\cdot$ then $\cdot$ else $\cdot$, with the expected types, and we restrict to models where they are interpreted as expected. We further assume that environments always contain the following declarations:

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

We require that models interpret all of these builtins in the natural way. In particular:

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

We generally omit type subscripts on the equality and quantifier symbols. With these operations, terms of type bool can be seen as formulas: we call them *local formulas* to distinguish them from the global formulas, which are the true formulas of our first-order logic. The semantics of a local formula is a random boolean variable, i.e. a function of η, ρ, but it evaluates all of its subformulas for the same values of η, ρ. In contrast, the semantics of a global formula is not parameterized by η, ρ, but these formulas express properties of random variables.

g) Names: As a last, but crucial, ingredient in our logic, we need a way to talk about specific random samplings. To do this, 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 $\tau_0 \to \tau_1$ where the *index* type τ_0 must be finite, which we write $\mathsf{finite}(\tau_0)$: concretely, $[\![\tau]\!]^{\eta}_{\mathbb{M}}$ must be finite for all η. We restrict to models where names have a specific interpretation: for any $\mathbb{M}$, there must 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 $\mathbb{M}$:

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

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 term whose free variables are either names other than n_1, or variables x such that $\mathsf{adv}(x)$ 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.

Example 4. *In order to model a keyed hash function, we would typically use types $\mathsf{message}$ and key and a function symbol $\mathsf{h} : \mathsf{message} \times \mathsf{key} \to \mathsf{message}$. Since keys are generally chosen*

randomly, they will be represented by names. Our notion of model does not force the sampling of keys to be uniform, i.e. we can reason over arbitrary key generation algorithms. However, it is reasonable to require that freshly sampled keys cannot be guessed by the attacker. This can be done by postulating the axiom $[\mathsf{k}\ \langle\rangle \neq t]$ *for any name* $\mathsf{k} : \mathsf{unit} \to \mathsf{key}$*, constant function symbol* $\langle\rangle : \mathsf{unit}$*, and term* t *that does not contain the name* k*. These conditions ensure that the random variables* $[\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$ *and* $[\![\mathsf{k}\ \langle\rangle]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$ *are independent; the probability that they coincide is thus negligible in models where keys are sampled uniformly enough in a large enough sample space.*

For convenience, we allow ourselves to use a name that has a base type, in which case the semantical assumptions on it are the same as if it were indexed over unit.

III. Expression, Games, and Adversaries

In this section, we introduce a programming language that we use to describe cryptographic games and adversaries. It is a simple While language extended with probabilistic random samplings, and a mechanism allowing adversaries to perform oracle calls. While adversaries are programs, not all programs describe valid adversaries: e.g. we will require that adversaries run in polynomial-time and properly use random samplings. Our programming language is tailored to simplify the connection with the logic in two ways: first, all the program randomness is sampled *eagerly*, before execution starts; second, the adversary is tasked with scheduling the oracles' randomness usage. Nonetheless, these non-standard features remain simple enough to ensure that our notion of adversaries (described using our programming language) is equivalent to the standard one.

A. Syntax

We assume a set of program variables $\mathcal{X}_\mathsf{p}$, and an intrinsic typing associating to each variable $v \in \mathcal{X}_\mathsf{p}$ a base type — we do not need a higher-order programming language. The *library* of our language, denoted by $\mathcal{L}_\mathsf{p}$, is a set of typed function symbols disjoint from $\mathcal{X}_\mathsf{p}$ representing built-in functions shared with the logic — i.e., we will have $\mathcal{L}_\mathsf{p} \subseteq \mathcal{E}$. We assume that $\mathcal{L}_\mathsf{p}$ contains at least the standard arithmetic and boolean operations (e.g. $0, \cdot + \cdot, \mathsf{if}\ \cdot\ \mathsf{then}\ \cdot\ \mathsf{else}\ \cdot, \mathsf{true}, \mathsf{false}$).

1) Expressions: We let Expr be the set of expressions built using $\mathcal{X}_\mathsf{p}$, $\mathcal{L}_\mathsf{p}$, and the special constant b, of type bool. This constant will denote the side we are in when describing a left or right cryptographic game. Formally, Expr is defined by:

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

Given an expression e, its *left projection* is the expression $e[\mathsf{b} \mapsto \mathsf{true}]$ obtained by substituting all occurrences of b by true. We define similarly the *right projection* $e[\mathsf{b} \mapsto \mathsf{false}]$ of e.

We assume a standard type system (whose rules we omit) and only consider well-typed expressions w.r.t. $\mathcal{X}_\mathsf{p}$ and $\mathcal{L}_\mathsf{p}$.

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

Fig. 1. Syntax of programs.

2) Tagged random samplings: The logic uses two different sources of randomness: the adversarial random tape ρ_a, given to logical adversaries (i.e. adversarial function symbols and the top-level distinguisher in the semantics of $\vec{u} \sim \vec{v}$) and the honest random tape ρ_h used to represent honest samplings performed by names. For programs, we need a more fined-grained separation of sources of randomness, as illustrated next.

Example 5. *Consider the indistinguishability formula* $\mathsf{att}(\mathsf{h}(\mathsf{m},\mathsf{k})) \sim \mathsf{att}(\mathsf{fresh})$ *with* m *and* k *distinct names and* att *an adversarial function. Under the PRF assumption on* h*, this formula is valid if we can build an adversary* $\mathcal{S}$ *against the PRF game of Example 1 such that the interaction of* $\mathcal{S}$ *with* G_0 *produces* $\mathsf{att}(\mathsf{h}(\mathsf{m},\mathsf{k}))$ *and the interaction of* $\mathcal{S}$ *with* G_1 *produces* $\mathsf{att}(\mathsf{fresh})$*. Here, such an adversary exists: it samples* m*, calls the real-or-random oracle on the result, and runs* att *on the output obtained from the oracle. In the execution of this simulator, it is useful to distinguish three kinds of randomness: the adversarial randomness used to run* att*, the randomness used by the simulator to sample* m*, and the randomness used by the oracle to sample* k*.*

To account for this, we use the notion of tagged random tapes, where each tag in $\mathtt{Tag} = \{\mathtt{T_A}, \mathtt{T_S}, \mathtt{T_G}\}$ represents a source of randomness used during the execution of the simulator $\mathcal{S}$,

- $\mathtt{T_A}$ is for (logical) adversarial randomness;
- $\mathtt{T_S}$ is for honest randomness that is directly sampled by $\mathcal{S}$;
- $\mathtt{T_G}$ is for the game (oracle) randomness that the simulator $\mathcal{S}$ cannot directly access.

The program $\mathcal{S}$ can directly sample from randomness tagged $\mathtt{T_A}$ or $\mathtt{T_S}$, but not randomness tagged $\mathtt{T_G}$. Intuitively, the logical random tape ρ_a corresponds to tag $\mathtt{T_A}$, while the logical random tape ρ_h must be partitioned between tags $\mathtt{T_S}$ and $\mathtt{T_G}$.

3) Programs: We consider a simple While programming language whose full syntax is given in Fig. 1. We require programs to be well-typed, but omit the obvious type system. Our language features the usual constructs — assignment, abort, conditional, while loop, ...— and two specific instructions used for random samplings and oracle calls, we described next.

The program $v \xleftarrow{\$} \mathtt{T}[e]$, where $v : \tau \in \mathcal{X}_\mathsf{p}$, $\mathtt{T} \in \mathtt{Tag}$, and e is of type int, samples a value of type τ using the randomness from random source $\mathtt{T}$ read at offset e, and stores it into v.

The program $v \leftarrow O_f(\vec{e})[\vec{e_g}; \vec{e_l}]$ 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 $\vec{e_g}, \vec{e_l}$ are type int. The integers $\vec{e_g}$ and $\vec{e_l}$ let the adversary control the offsets at which the oracle reads its randomness — the attacker controls the

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

Fig. 2. Syntax of games defined over oracle names $\mathcal{O}$.

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.

Example 6. *Assuming $n, m, s : \mathsf{message} \in \mathcal{X}_{\mathsf{p}}$, the program*

$$n \xleftarrow{\$} \mathtt{T_S}[1];\ m \xleftarrow{\$} \mathtt{T_S}[2];\ s \xleftarrow{\$} \mathtt{T_S}[3];\ w \leftarrow O_{\mathrm{hash}}(n)[7;\cdot];$$
$$w' \leftarrow O_{\textit{real-or-random}}(\mathsf{h}(m,s))[7;3]$$

may represent an adversary against the PRF game of Example 1. It directly samples three values of type message *at offsets 1, 2 and 3 in the honest random tape, corresponding to tag $\mathtt{T_S}$. It calls the hash oracle on the first sampled value, indicating that the game's key must be read from the game tape for type* message *at offset 7. It computes its own hash of the second sampled value, keyed by the third sampled value, and finally calls the real-or-random oracle on this hash, indicating the same global offset for the key, and a local offset of 3 for the oracle's randomness. The global and local samplings of the oracle calls are done using tag $\mathtt{T_G}$.*

4) Games: Cryptographic games set up some data (e.g. randomly sampling keys) and provide functionalities through *oracles* to compute over this data. Oracles may modify the game's memory. As explained before, we are interested in pairs of games $(\mathsf{G}_0, \mathsf{G}_1)$ that are assumed to be indistinguishable. As for expressions, we factorize their common behavior by defining a single game G that can use the boolean variable b, such that G equals G_i 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 *whose syntax is given in Fig. 2. Declarations contain, in order, sequences of:*

1) initialisation of variables, either:

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

2) $f(\vec{v}) := \{v_1^l \xleftarrow{\$}; \ldots; v_k^l \xleftarrow{\$}; \mathsf{p};$ return $e\}$, which defines an oracle $f \in \mathcal{O}$ with inputs $\vec{v}$ that initializes a sequence of local *random variables $v_1^l, \ldots, v_k^l$, then executes a program* p *without random samplings nor oracle calls, 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 $f \in \mathcal{O}$. Given one such definition, we let:

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

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

For the rest of the article, we let G be an arbitrary but fixed game, well-typed w.r.t. $\mathcal{X}_{\mathsf{p}}$ and $\mathcal{L}_{\mathsf{p}}$.

Example 7. *We define the PRF game over $\mathcal{O} = \{\mathsf{hash}, \mathsf{chal}\}$:*

$$
\begin{array}{l}
k \xleftarrow{\$};\ \ell_{\mathsf{hash}} \leftarrow [];\ \ell_{\mathsf{chal}} \leftarrow []; \\
\mathsf{hash}(x) := \{\ \ell_{\mathsf{hash}} \leftarrow x :: \ell_{\mathsf{hash}}; \\
\qquad \textbf{return } (\text{if } x \notin \ell_{\mathsf{chal}} \text{ then } \mathsf{h}(x,k) \text{ else zero})\ \} \\
\mathsf{chal}(x) := \{\ r \xleftarrow{\$}; \\
\qquad v \leftarrow \text{if } x \notin \ell_{\mathsf{hash}} \cup \ell_{\mathsf{chal}} \text{ then} \\
\qquad\qquad \text{if b then } \mathsf{h}(x,k) \text{ else } r \\
\qquad\quad \text{else zero}; \\
\qquad \ell_{\mathsf{chal}} \leftarrow x :: \ell_{\mathsf{chal}}; \\
\qquad \textbf{return } v\ \}
\end{array}
$$

We use library functions for the empty list $[]$ and list concatenation $\cup$, and the if-then-else — the conditional above is not at the program level. Our game generalize the one in Example 1 by allowing multiple calls to the challenge, which in turn requires that we use two logs ℓ_{hash} and ℓ_{chal} to avoid that a challenge message is later used for the hash oracle. The game features a globally sampled variable k and a locally sampled variable r in the challenge oracle. The nested conditional in that challenge uses the special variable b *to specify the different behaviors of G_0 and G_1.*

Anticipating on what follows, the program from Example 6 will execute against our PRF game. After the execution with $\mathsf{b} = 0$ and $\mathsf{b} = 1$ respectively, the evaluation of $\langle w, w' \rangle$ will yield random variables with the same distributions as:

$$\langle \mathsf{h(n,k)}, \text{if } \mathsf{n} \neq \mathsf{m} \text{ then } \mathsf{h(h(m,s),k)} \rangle$$
$$\text{and} \quad \langle \mathsf{h(n,k)}, \text{if } \mathsf{n} \neq \mathsf{m} \text{ then fresh} \rangle.$$

Intuitively, the indistinguishability of the PRF game G thus implies the following logical indistinguishability, where we simplified away the overwhelmingly true name dis-equality:

$$\mathsf{h(n,k)}, \mathsf{h(h(m,s)k)} \sim \mathsf{h(n,k)}, \mathsf{fresh}.$$

B. Semantics

The semantics of expressions and programs is parameterized by a logical model $\mathbb{M}$ of a (logical) environment $\mathcal{E}$. The model $\mathbb{M}$ is used to specify the semantics of all library functions in $\mathcal{L}_{\mathsf{p}}$. To that end, we require that $\mathcal{E}$ is *compatible* with $\mathcal{X}_{\mathsf{p}}$ and $\mathcal{L}_{\mathsf{p}}$, in the following sense: $\mathcal{L}_{\mathsf{p}} \subseteq \mathcal{E}$ and the set of variables defined or declared in $\mathcal{E}$ is disjoint from $\mathcal{X}_{\mathsf{p}}$.

1) Memories: A *memory* $\mu \in \mathsf{Mem}_{\mathbb{M},\eta}$ w.r.t. a type structure $\mathbb{M}$ and η is a function μ 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 $v' \neq v$.

2) 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 $\mathsf{R}_{\mathbb{M},\eta}(\tau_b)$ bits from the random tapes, and then use the sampling algorithm $[\![\tau_b]\!]^{\$}_{\mathbb{M}}(\eta, w_\$)$ provided by the model to obtain a value in $[\![\tau_b]\!]^{\eta}_{\mathbb{M}}$. To simplify the presentation and analysis of the bi-deduction logic in Section IV, we use a different random tape for each usage: we will use a family of random tapes, one for each pair $(\mathtt{T}, \tau_b)$ of randomness source (i.e. tag $\mathtt{T} \in \{\mathtt{T_A}, \mathtt{T_G}, \mathtt{T_S}\}$) and base type $\tau_b \in \mathbb{B}$ we are sampling from. However, we only consider bool for $\mathtt{T_A}$ since adversarial randomness is only needed for the adversarial function symbols in $\mathcal{L}_\mathsf{p}$.

Definition 2. *A* program random tape $\mathfrak{p}$ *is a family* $(\mathfrak{p}|_L)_{L\in\mathcal{S}}$ *of infinite sequences of bits indexed by the set of* labels*:*

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

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.

3) Semantics: The semantics $[e]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu}$ of an expression e of type τ is a value in $[\![\tau]\!]^{\eta}_{\mathbb{M}}$. 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 $(\!|\mathsf{p}|\!)^{\eta,\mathfrak{p}}_{\mathsf{G},\mathbb{M},i,\mu} \in \mathsf{Mem}_{\mathbb{M},\eta} \cup \{\bot\}$ of a program p 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 sack of conciseness, we write $[e]^{\eta,\mathfrak{p}}_{\mu}$ (resp. $(\!|\mathsf{p}|\!)^{\eta,\mathfrak{p}}_{\mu}$) when $\mathbb{M}$, i and G are clear from context.

The definitions of these semantics is as expected from the above explanations. We show below one interesting case, providing the full definitions are in Appendix B:

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

where $k = [e]^{\eta,\mathfrak{p}}_{\mu}$ and v has type τ.

We use a special variable res to store the return value of a program. With this convention, we define the probability $\Pr_\mathfrak{p}((\!|\mathsf{p}|\!)^{\eta,\mathfrak{p}}_{\mathsf{G},\mathbb{M},i,\mu})$ that a program p terminates in an accepting state, starting in a memory μ with security parameter η and interacting with side $i \in \{0,1\}$ of G, as follows:

$$\Pr_\mathfrak{p}((\!|\mathsf{p}|\!)^{\eta,\mathfrak{p}}_{\mathsf{G},\mathbb{M},i,\mu}) \stackrel{\text{def}}{=} \Pr_\mathfrak{p}\left[\mu'[\mathsf{res}] = 1 \text{ where } \mu' = (\!|\mathsf{p}|\!)^{\eta,\mathfrak{p}}_{\mathsf{G},\mathbb{M},i,\mu}\right]$$

4) Cost model: In order to define what an adversary is, we need to restrict ourselves to PTIME programs, which requires a time cost model. Rather than providing an explicit cost model, which would be tedious and possibly unnecessarily restrictive, we assume an arbitrary cost model satisfying a few expected and standard properties which we detail in Appendix B-C.

5) Security of a game: An adversary against a game is a PTIME program — w.r.t. our cost model — that respect the game's execution, i.e. that: does not read nor write the game internal variables nor the special side constant b; does not read the game tapes (labeled with $\mathtt{T_G}$); and properly provides randomness offsets to the game: local offsets must be fresh, and global offsets must be consistent across oracle calls. For more details, see Appendix B-E.

We finally define what it means for a game to be secure.

Definition 3. *A game* G *is secure in a compatible model* $\mathbb{M}$ *if for any adversary* p*, the following quantity is negligible in* η*:*

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

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

Example 8. *The game of Example 7 is secure in a model $\mathbb{M}$ iff. the standard cryptographic PRF assumption is satisfied for the implementation of h in $\mathbb{M}$.*

IV. Bi-deduction

Now that we have fixed our notions of games and adversaries, we develop the central concept of this work: bi-deduction in presence of a cryptographic game. When working with bi-deduction, we will deal with several pairs of objects, where each component is involved in the deduction on one side $i \in \{0,1\}$. We introduce special notations for such pairs, following the style of [8].

Definition 4 (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 $\#$ downward. E.g. $f(\#(u,v), g(\#(s,t)))$ denotes $\#(f(u, g(s)), f(v, g(t)))$.

Our end goal in this work is to develop a method for deriving indistinguishabilities from cryptographic assumptions expressed as games, which should in particular subsume existing cryptographic axioms. To this end, we shall significantly generalize the notion of bi-deduction from [19]. In that work, a bi-deduction $u_\# \rhd v_\#$ holds when there exists a deterministic PTIME simulator which, for each $i \in \{0,1\}$, computes v_i when given u_i as input. This can be used to support useful inferences on indistinguishability, as shown in the next rule:

$$\frac{\vec{u_0} \sim \vec{u_1} \quad \#(\vec{u_0}, \vec{u_1}) \rhd \#(\vec{v_0}, \vec{v_1})}{\vec{v_0} \sim \vec{v_1}}$$

To prove the soundness of this rule, assume the contrary: then the PTIME distinguisher $\mathcal{D}$ against $\vec{v_0} \sim \vec{v_1}$ can be composed with the simulator $\mathcal{S}$ witnessing $\#(\vec{u_0}, \vec{u_1}) \rhd \#(\vec{v_0}, \vec{v_1})$ to obtain a PTIME distinguisher $\mathcal{D} \circ \mathcal{S}$ against $\vec{u_0} \sim \vec{u_1}$.

We seek to leverage a similar argument to establish that $\vec{v_0} \sim \vec{v_1}$ is a consequence of the security of a cryptographic

game G. This will be expressed[1] as $\#(\emptyset,\emptyset) \rhd \#(\vec{v_0},\vec{v_1})$ but with a stronger notion of computation underlying bi-deduction: to exploit the full generality of the argument, we will consider arbitrary G-adversaries as simulators. This will lead to a generalized bi-deduction judgement involving constraints on the usage of random tapes, and assertions for describing the game's memory at a point of the simulator's computation. We present next these key ingredients, before introducing bi-deduction itself and a proof system for it.

A. 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, an adversary 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). Having the simulator sample k itself would be unsound, as an adversary is forbidden from accessing the game's random samplings directly.

To avoid such issues, 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:

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

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

Definition 5. *A* name constraint *is a tuple* $c = (\vec{\alpha}, \mathsf{n}, t, T, f)$ *where* $\vec{\alpha}$ *is a vector of variables in* $\mathcal{X}$, n *is a name,* t *is a term,* $T \in \mathsf{TAG}_{\mathit{constr}}$*, and* f *is a local formula. A* constraint system $\mathcal{C}$ *is a list of name constraints.*

Intuitively, the constraint above expresses that, for an arbitrary number of instantiations 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 $\vec{\alpha}$.

Example 9. $[(\{i\}, \mathsf{n}, i, \mathtt{T}^{\mathsf{glob}}_{\mathsf{G},v}, f),\ (\{i\}, \mathsf{n}, i, \mathtt{T}_S, f')]$ *is a constraint system that expresses that: for every value of* i *for which* f *holds, the name* n i *is used to represent the global sampling of variable* v *of the game; for every value of* i *for which* f' *holds, the name* n i *represents a sampling performed by the simulator. For this to make sense, we expect* f *and* f' *to be mutually exclusive.*

[1]Although we will initially be concerned with judgements without any input, i.e. $\vec{u_\#} = \emptyset$, the general form $\vec{u_\#} \rhd \vec{v_\#}$ will still be useful, allowing e.g. transitivity and induction rules in our proof system. Also note that we only use our generalized bi-deduction to establish indistinguishabilities; $\vec{u_\#} \rhd \vec{v_\#}$ cannot be used to establish that $\vec{v_0} \sim \vec{v_1}$ follows from $\vec{u_0} \sim \vec{u_1}$ as in [19].

To reflect the intended semantics of constraints we define $\mathcal{N}^{\eta,\rho}_{\mathcal{C},\mathbb{M}} \stackrel{\mathsf{def}}{=} \bigcup_{c\in\mathcal{C}} \mathcal{N}^{\eta,\rho}_{c,\mathbb{M}}$ where:

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

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

Intuitively, a constraint system expresses how the computation underlying a bi-deduction judgement uses names. We define a validity criterion for constraint systems that captures when the usage of names is consistent, which is necessary to ensure that the computation does correspond to an adversary. Validity consists of three conditions:

- name-tag associations must be *functional*: no name is associated to two different tags;
- the local samplings must be *fresh*: the associated names do not occur anywhere else;
- a globally sampled variable must be associated to a *unique* name.

These conditions must hold for every η, ρ such that the conditions f hold, and are defined formally in Fig. 3.

We define the *validity* of a constraint system $\mathcal{C}$ as the conjunction of all conditions on all pairs of constraint occurrences:

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

As expected, $\Theta \models \mathrm{Valid}(\mathcal{C}')$ and $\Theta \models \mathcal{C} \subseteq \mathcal{C}'$ imply $\Theta \models \mathrm{Valid}(\mathcal{C})$. The validity condition relies on the exact truth predicate, in other words we require our attackers 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, as constraints on random samplings are intentional: we certainly do not rule out the possibility that an attacker, performing a random sampling by itself, happens to obtain the same value as a game's random sampling.

Later, we will make use of bi-systems of constraints $\mathcal{C}_\#$. In practice, they will be pairs of lists of the same length, so we view and manipulate them as lists of bi-constraints. We define $\mathrm{Valid}(\mathcal{C}_\#)$ as $\mathrm{Valid}(\mathcal{C}_1) \barwedge \mathrm{Valid}(\mathcal{C}_2)$.

B. Assertion Logic

Since the values returned by oracles may depend on the game's global state, we need to keep track of this state during the adversarial computation underlying bi-deduction.

Example 10. *In the PRF game, if* k *corresponds to the game's globally sampled key,* fresh *to a local sampling of the challenge oracle, and* m *is an arbitrary adversarial message, we can bi-deduce* $(\emptyset,\emptyset) \rhd \#(\mathsf{h}(m,\mathsf{k}),\mathsf{fresh})$*: the simulator simply calls the challenge oracle on* m*. However, we should* not *have* $(\emptyset,\emptyset) \rhd h(m,\mathsf{k}), \#(\mathsf{h}(m,\mathsf{k}),\mathsf{fresh})$*: to compute these bi-terms, the simulator would have to call both the hash and the challenge oracles on* m*, and the second call would fail because it would find* m *in either* ℓ_{chal} *or* ℓ_{hash}*. However,*

$$\mathrm{Fun}(c_1,c_2) \stackrel{\mathsf{def}}{=} \forall\vec{\alpha_1}\forall\vec{\alpha_2}.\ f_1 \wedge f_2 \Rightarrow t_1 \neq t_2 \quad \text{when } T_1 \neq T_2 \text{ and } n_1 = n_2, \quad \text{and } \mathrm{Fun}(c_1,c_2) \stackrel{\mathsf{def}}{=} \top \text{ otherwise}$$

$$\mathrm{Fresh}(c_1,c_2) \stackrel{\mathsf{def}}{=} \forall\vec{\alpha_1}\forall\vec{\alpha_2}.\ f_1 \wedge f_2 \Rightarrow c_1(\vec{\alpha_1}) \neq c_2(\vec{\alpha_2}) \Rightarrow t_1 \neq t_2 \quad \text{when } T_1 = T_2 = \mathtt{T}^{\mathsf{loc}}_{\mathsf{G}},\ n_1 = n_2, \quad \text{and } \mathrm{Fun}(c_1,c_2) \stackrel{\mathsf{def}}{=} \top \text{ otherwise}$$

$$\mathrm{Unique}(c_1,c_2) \stackrel{\mathsf{def}}{=} \begin{cases} \forall\vec{\alpha_1}\forall\vec{\alpha_2}.\ f_1 \wedge f_2 \Rightarrow t_1 = t_2 & \text{when } T_1 = T_2 \in \{\mathtt{T}^{\mathsf{glob}}_{\mathsf{G},v} \mid v \in \mathsf{G.gs}\},\ n_1 = n_2 \\ \forall\vec{\alpha_1}\forall\vec{\alpha_2}.\ f_1 \wedge f_2 \Rightarrow \bot & \text{when } T_1 = T_2 \in \{\mathtt{T}^{\mathsf{glob}}_{\mathsf{G},v} \mid v \in \mathsf{G.gs}\},\ n_1 \neq n_2 \\ \top & \text{otherwise} \end{cases}$$

where $c_1 = (\vec{\alpha_1}, \mathsf{n}_1, t_1, T_1, f_1)$ and $c_2 = (\vec{\alpha_2}, \mathsf{n}_2, t_2, T_2, f_2)$; and where we write $c_1(\vec{\alpha_1}) \neq c_2(\vec{\alpha_2})$ as a shorthand for $\top$ if c_1 and c_2 are distinct occurrences, and $\vec{\alpha_1} \neq \vec{\alpha_2}$ otherwise.

Fig. 3. Constraint validity conditions.

we should have $(\emptyset,\emptyset) \rhd h(n,\mathsf{k}), \#(\mathsf{h}(m,\mathsf{k}),\mathsf{fresh})$ *for any adversarial message* n *always distinct from* m*, i.e.* $[n \neq m]_{\mathrm{e}}$.

A natural way to control the game's state through bi-deduction is to enrich our judgement with pre- and post-conditions. For now we shall rely on an abstract assertion language — a concrete instance of it will be considered later.

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. Adding $\mathfrak{p}$ to the satisfaction relation would not be useful, as relevant samplings by the program and oracles can be accessed directly in the memory.

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 $\mathbb{M},\eta,\rho,\mu \models^A \psi$.

Example 11. *To see why local formulas are useful in assertions, let us elaborate on Example 10. When* n *and* m *are two names* n *and* m*, we cannot guarantee that they are* always *distinct. However, we have the following:*

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

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.*

C. Bi-deduction Judgement

We now have all the ingredients to form our bi-deduction judgement. Defining its semantics, though, requires a little more work. We start with a basic notion of computation, on which we will elaborate.

Definition 6. *Let* $u^1_\#,\dots,u^m_\#,v_\#$ *be a sequence of bi-terms well-typed in some environment* $\mathcal{E}$*, which all have base types. We say that a program* p *with distinguished variables* $X_1,\dots,X_m$ *and* res computes $\vec{u_\#} \rhd v_\#$ *w.r.t.* $\mathbb{M},\eta,\mathfrak{p},\rho,\mu$ *and side* $i \in \{0,1\}$ *when:*

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

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

a) Relating logical and program tapes: Naively, one may then say that a bi-deduction $u_\# \rhd v_\#$ holds w.r.t. a game G when there exists an adversary p against G which computes $u_\# \rhd v_\#$ 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*. These parts will roughly be captured by $\mathcal{N}^{\eta,\rho}_{\mathbb{M}:\mathcal{E},\mathcal{C}}$ where $\mathcal{C}$ is the judgement's constraints system.

Example 12. *In the PRF game, if we need a name* k *to correspond to the 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* $\mathfrak{p}$*).*

We now turn to defining the relation between logical and program random tapes which is associated to a constraint system. To do so, we assume a fixed but arbitrary 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 $O_{\mathbb{M},\eta}$ is injective and $(1^\eta,a) \mapsto O_{\mathbb{M},\eta}(\mathsf{n},a)$ is PTIME computable.

Definition 7. *Let* $\mathcal{C}$ *be a constraint system and* $\mathbb{M}$ *a model, both w.r.t.* $\mathcal{E}$*. For any* $\eta \in \mathbb{N}$*, we define* $\mathcal{R}^\eta_{\mathcal{C},\mathbb{M}}$ *as the relation between* $\mathbb{T}_{\mathbb{M},\eta}$ *and program random tapes* $\mathfrak{P}$ *such that* $\rho\ \mathcal{R}^\eta_{\mathcal{C},\mathbb{M}}\ \mathfrak{p}$ *holds when* ρ_{a} *is a prefix of* $\mathfrak{p}[\mathtt{T}_{\mathbb{A}}, \mathsf{bool}]$ *and for all* $(\mathsf{n},a,\mathtt{T}) \in \mathcal{N}^{\eta,\rho}_{\mathcal{C},\mathbb{M}}$*,* $[\![\mathsf{n}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(a) = \mathfrak{p}|^\eta_{\mathtt{T}}[O_{\mathbb{M},\eta}(\mathsf{n},a)]$.

Intuitively, the relation $\mathcal{R}^\eta_{\mathcal{C},\mathbb{M}}$ relates pairs of logical and program tapes that coincide on segments identified by $\mathcal{C}$.

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

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

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:

$$\Pr_{\mathfrak{p}\in\mathfrak{P}}\left((\!|p|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},i}[\mathsf{res}]=x\right)=\Pr_{\rho\in\mathbb{T}_{\mathbb{M},\eta}}\left([\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}=x\right)\quad(1)$$

we rely on the standard notions of probabilistic coupling and lifting (as in [26]). We only present the main intuitions here, details are postponed to Appendix C-C. 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 marginal follow the same distributions as the distributions endowing, resp., $\mathbb{T}_{\mathbb{M},\eta}$ and $\mathfrak{P}$. E.g., $\mathbb{C}:\mathbb{T}_{\mathbb{M},\eta}\bowtie\mathfrak{P}$ implies that for any x:

$$\Pr_{\rho\in\mathbb{T}_{\mathbb{M},\eta}}\left([\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}=x\right)=\Pr_{(\rho,\mathfrak{p})\in\mathbb{C}}\left([\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}=x\right)\quad(2)$$

$$\Pr_{\mathfrak{p}\in\mathfrak{P}}\left((\!|p|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},i}[\mathsf{res}]=x\right)=\Pr_{(\rho,\mathfrak{p})\in\mathbb{C}}\left((\!|p|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},i}[\mathsf{res}]=x\right)\quad(3)$$

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 $\mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}$ (roughly, this means that $\mathbb{C}$ only samples pairs of tapes related by $\mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}$). Then Eq. (1) holds. Indeed:

$$\begin{aligned}
&\Pr_{\rho\in\mathbb{T}_{\mathbb{M},\eta}}\left([\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}=x\right)\\
&=\Pr_{(\rho,\mathfrak{p})\in\mathbb{C}}\left([\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}=x\right) && \text{(Eq. (2))}\\
&=\Pr_{(\rho,\mathfrak{p})\in\mathbb{C}}\left((\!|p|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},i}[\mathsf{res}]=x\right) && (\mathbb{C}\text{ contained in }\mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}})\\
&=\Pr_{\mathfrak{p}\in\mathfrak{P}}\left((\!|p|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},i}[\mathsf{res}]=x\right) && \text{(Eq. (3))}
\end{aligned}$$

c) Couplings from constraint systems: Given a constraint system $\mathcal{C}$, we would like to build a coupling that is contained in $\mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}$. It turns out that this cannot always be achieved: counter-examples arise when a constraint $c=(\vec{\alpha},\mathsf{n},t,\mathtt{T},f)$ features a condition f (or an index t) that depends on the name n t introduced in the constraint (see Example 15, Appendix C for an explicit counter-example). Such pathological cases are of no use for our use of constraint systems, and we rule them out by introducing in Appendix C-C the 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, following the order in which constraints appear in $\mathcal{C}$: when processing a constraint c, we are able to compute f and t using the already sampled parts of the tape, by well-formedness; then, if f holds, we sample the segments of the logical and program tapes determined by n, t and $\mathtt{T}$ (validity ensures that these segments are not yet sampled). Once all constraints are processes, the rest of the tapes in sampled using to relevant probability distributions.

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

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

d) Bi-deduction: We finally define bi-deduction:

Definition 8 (Bi-deduction judgement and semantics)**.** *A bi-deduction judgement is of the form:*

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

where G is a game, $\mathcal{E}$ is an environment, Θ is a set of global formulas, $\mathcal{C}_{\#}$ 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 G-adversary p such that for any model $\mathbb{M}:\mathcal{E}$ extending $\mathbb{M}_0$ in such a way that $\mathbb{M}\models\Theta\tilde{\wedge}\mathrm{Valid}(\mathcal{C}_{\#})$, for any $\eta\in\mathbb{N}$, $i\in\{0,1\}$, $\mathcal{C}_{\#}$ is well-formed w.r.t. $\mathbb{M},\eta$ and for any tapes ρ $\mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}$ $\mathfrak{p}$, 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 $L_{\$}$:*

$$\begin{aligned}
G_{\$}&\subseteq\{\,O_{\mathbb{M},\eta}(\mathsf{n},a)\mid\langle\mathsf{n},a,\mathtt{T}^{\mathsf{glob}}_{\mathsf{G},v}\rangle\in\mathcal{N}^{\eta,\rho}_{c,\mathbb{M}},\ c\in\mathcal{C}\,\}\\
\text{and}\quad L_{\$}&\subseteq\{\,O_{\mathbb{M},\eta}(\mathsf{n},a)\mid\langle\mathsf{n},a,\mathtt{T}^{\mathsf{loc}}_{\mathsf{G}}\rangle\in\mathcal{N}^{\eta,\rho}_{c,\mathbb{M}},\ c\in\mathcal{C}\,\}.
\end{aligned}$$

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 G-adversaries.

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 $\mathcal{C}_{\#}$, $\vec{v_{\#}}$ and $\psi_{\#}$:*

$$\textsc{Bi-deduce}\quad\frac{\mathcal{E},\Theta\vdash\mathrm{Valid}(\mathcal{C}_{\#})\qquad\mathcal{E},\Theta,\mathcal{C}_{\#},(\varphi_{\#},\psi_{\#})\vdash\emptyset\rhd_{\mathsf{G}}\vec{v_{\#}}}{\mathcal{E},\Theta\vdash\vec{v_0}\sim\vec{v_1}}$$

The proof is given in Appendix D-C.

D. 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 bi-deduced. To enable expressive rules, it is useful to consider vector of conditional terms. We will thus consider bi-deductions of the form $\vec{u}\rhd(f_1,\text{if }f_1\text{ then }t_1,\ldots,f_n,\text{if }f_n\text{ then }t_n)$, 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 make use of two operations on constraint systems. First, the concatenation of bi-constraint systems is defined as $\#(\mathcal{C}_0^1,\mathcal{C}_1^1)\cdot\#(\mathcal{C}_0^2,\mathcal{C}_1^2)=\#(\mathcal{C}_0^1\cdot\mathcal{C}_0^2,\mathcal{C}_1^1\cdot\mathcal{C}_1^2)$. Note that the validity of the concatenation of two constraints systems

carries over to each of them and the well-formedness of two constraints systems carries over to their concatenation. Second, we define the generalization $\forall x.\mathcal{C}$ of $\mathcal{C}$ over x as the constraint system $\mathcal{C}$ where x is added to the vector of bound variables in all basic constraints. The validity of $\forall x.\mathcal{C}$ implies that of $\mathcal{C}$, and the well-formedness of $\mathcal{C}$ carries on to its generalizations.

We show a selected set of rules in Fig. 4, which we describe below. First, our proof system features rules for basic simulator constructions — e.g. REFL, FA, DROP, and IF-THEN-ELSE — as well as weakening rules, e.g. WEAK.COND for term conditions and WEAK.CONSTR for constraints. 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 $\mathrm{Valid}(\mathcal{C}^1_\# \cdot \mathcal{C}^2_\#)$, 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 $\mathcal{C}^1_\# \cdot \mathcal{C}^2_\#$ 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.

In order to represent unbounded collections of objects to bi-deduce, we extend the bi-deduction judgement beyond terms of base type, allowing order-1 types when the argument types are enumerable. This can be done without changing the semantics of bi-deduction, simply viewing these functions as an explicit representation of their graph. This extension notably brings the LAMBDA and INDUCTION rules of Fig. 4, 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, $\mathsf{well\text{-}founded}_\tau(<)$ states that $(\llbracket\tau\rrbracket^\eta_{\mathbb{M}}, \llbracket<\rrbracket^\eta_{\mathbb{M},\mathcal{E}})$ is well-founded (details in Appendix D-A).

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. 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 Section D-B3).

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:*

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

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

V. PROOF SEARCH AND IMPLEMENTATION

We now describe the specification, heuristic and design choices of our proof-search procedure — called $\mathrm{search}_\rhd$ — for bi-deduction, as well as its implementation in our extension of SQUIRREL [25], now merged into the main branch of the tool. The tool now allows users to specify arbitrary games, and bi-deduction verification is made available to the users through a tactic **crypto** based on $\mathrm{search}_\rhd$, which takes as input the game 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.

1) Scope: Our goal is for $\mathrm{search}_\rhd$ 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 to proof-goal using its indistinguishability logic [20] 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: $\mathrm{search}_\rhd$ 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).

2) Proof-search: $\mathrm{search}_\rhd$ 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, a bi-constraints system $\mathcal{C}_\#$, a pre-condition $\varphi_\#$, inputs $\vec{u_\#}$ and outputs $\vec{v_\#}$, $\mathrm{search}_\rhd$ looks for additional hypotheses Θ', constraints $\mathcal{C}'_\#$ and a post-condition $\psi_\#$, such that:

$$\mathcal{E}, \Theta \cup \Theta', \mathcal{C}_\# \cdot \mathcal{C}'_\#, (\varphi_\#, \psi_\#) \vdash \vec{u_\#} \rhd \vec{v_\#} \quad \text{is valid.}$$

It proceeds by a backward proof search, applying rules whose conclusion matches $\vec{v_\#}$ and recursing on the rule premises.

3) Heuristic: $\mathrm{search}_\rhd$ tries to greedily apply the oracle rule, but avoids using it in scenarios where name constraints added by the oracle rule would trivially lead to a unsatisfiable constraint system. E.g., $\mathrm{search}_\rhd$ will not apply the oracle rule if doing so would associate a name with a global tag $\mathrm{T}^{\mathsf{glob}}_{\mathsf{G},v}$ when this tag is already associated to a different name. When the status of a name wrt. constraints does not necessarily allow to use an oracle, $\mathrm{search}_\rhd$ 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

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

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

$$\textsc{Transitivity}\ \frac{\mathcal{E},\Theta,\mathcal{C}^1_\#,(\varphi_\#,\varphi'_\#)\vdash \vec{u_\#}\rhd\vec{t_\#}\quad \mathcal{E},\Theta,\mathcal{C}^2_\#,(\varphi'_\#,\psi_\#)\vdash \vec{u_\#},\vec{t_\#}\rhd\vec{v_\#}}{\mathcal{E},\Theta,\mathcal{C}^1_\#\cdot\mathcal{C}^2_\#,(\varphi_\#,\psi_\#)\vdash \vec{u_\#}\rhd\vec{t_\#},\vec{v_\#}}$$

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

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

$$\textsc{Drop}\ \frac{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#)\vdash \vec{u_\#}\rhd\vec{v_\#},\vec{t_\#}}{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#)\vdash \vec{u_\#}\rhd\vec{v_\#}}$$

$$\textsc{If-then-else}\ \frac{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#)\vdash \vec{u_\#}\rhd \vec{v_\#},(b_\#\mid f_\#),(t_\#\mid f_\#\wedge b_\#),(t'_\#\mid f_\#\wedge\neg b_\#)}{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#)\vdash \vec{u_\#}\rhd\vec{v_\#},(\mathsf{if}\ b_\#\ \mathsf{then}\ t_\#\ \mathsf{else}\ t'_\#\mid f_\#)}$$

$$\textsc{Lambda}\ \frac{(\mathcal{E},x:\tau),\Theta,\mathcal{C}_\#,(\varphi_\#,\varphi_\#)\vdash \vec{u_\#}\rhd(t_\#\mid f_\#)\quad \mathcal{E},x:\tau\vdash t_\#:\tau_{\mathsf{b}}\quad \mathsf{enum}(\tau)}{\mathcal{E},\Theta,\forall(x:\tau).\mathcal{C}_\#,(\varphi_\#,\varphi_\#)\vdash \vec{u_\#}\rhd(\lambda(x:\tau).t_\#\mid f_\#)}$$

$$\textsc{Induction}\ \frac{(\mathcal{E},x:\tau),\Theta,\mathcal{C}_\#,(\varphi_\#,\varphi_\#)\vdash \vec{u_\#},(\lambda(y:\tau).\mathsf{if}\ y<x\ \mathsf{then}\ t[x\mapsto y]\mid f_\#)\rhd(t_\#\mid f_\#)\quad \mathsf{finite}(\tau)\quad \mathsf{fixed}(\tau)\quad \mathcal{E},\Theta\vdash \mathsf{well\text{-}founded}_\tau(<)\ \tilde{\wedge}\ \mathsf{adv}(<)}{\mathcal{E},\Theta,\forall(x:\tau).\mathcal{C}_\#,(\varphi_\#,\varphi_\#)\vdash \vec{u_\#}\rhd(\lambda(x:\tau).t_\#\mid f_\#)}$$

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

Fig. 4. Selected set of rules.

$\mathsf{h}(m,\mathsf{k}\ j)$ 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 $j\neq i$).

4) Implementation: In addition to an implementation of search$_\rhd$, the new **crypto** tactic brings a new syntax to declare arbitrary game, 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 Example 7. 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 be called on a equivalence $\vec{v_0}\sim\vec{v_1}$, **crypto** 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-A for more details on how recursion is handled.

VI. 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 [25] (in sub-directory `examples/crypto/`), as well as in HTML files that allow to replay the run 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 [27] using a previously unsupported cryptographic assumption: the CCA$_\$$ game that 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 [28], which crucially relies on the CCA$_2$ game that was previously unsupported in Squirrel.

All cryptographic tactics in Squirrel can only handle $\#(\_,\_)$ in the indistinguishability to prove, and not in the protocol being used. Interestingly, **crypto** does not have this limitation: `global-cpa.sp` shows that we can 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.

VII. 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 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 crypto assumptions, e.g. from electronic voting. From a theoretical point of view, our bi-deduction relies on an exact semantics which complicates some proof rules; we will explore several ways to relax this limitation.

References

[1] "CVE-2014-0160 aka. the Heartbleed bug." Available from MITRE, 2013. [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160

[2] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1–19.

[3] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann, "Imperfect forward secrecy: How Diffie-Hellman fails in practice," in *22nd ACM Conference on Computer and Communications Security*, Oct. 2015.

[4] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, G. Pellegrino, and A. Sorniotti, "An authentication flaw in browser-based single sign-on protocols: Impact and remediations," *Computers & Security*, vol. 33, pp. 41–58, 2013.

[5] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the TLS 1.3 standard candidate," in *2017 IEEE Symposium on Security and Privacy*. IEEE, 2017, pp. 483–502.

[6] M. Barbosa, G. Barthe, K. Bhargavan, B. Blanchet, C. Cremers, K. Liao, and B. Parno, "Sok: Computer-aided cryptography," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 777–795.

[7] V. Shoup, "Sequences of games: a tool for taming complexity in security proofs," *IACR Cryptol. ePrint Arch.*, p. 332, 2004.

[8] B. Blanchet, "A computationally sound mechanized prover for security protocols," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2006, pp. 140–154.

[9] J. Gancher, S. Gibson, P. Singh, S. Dharanikota, and B. Parno, "Owl: Compositional verification of security protocols via an information-flow type system," in *2023 2023 IEEE Symposium on Security and Privacy (SP) (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2023, pp. 1130–1147. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.00166

[10] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin, "Computer-aided security proofs for the working cryptographer," in *CRYPTO*, ser. Lecture Notes in Computer Science, vol. 6841. Springer, 2011, pp. 71–90.

[11] D. A. Basin, A. Lochbihler, and S. R. Sefidgar, "Crypthol: Game-based proofs in higher-order logic," *J. Cryptol.*, vol. 33, no. 2, pp. 494–566, 2020.

[12] C. Abate, P. G. Haselwarter, E. Rivas, A. V. Muylder, T. Winterhalter, C. Hritcu, K. Maillard, and B. Spitters, "Ssprove: A foundational framework for modular cryptographic proofs in coq," in *CSF*. IEEE, 2021, pp. 1–15.

[13] G. Bana and H. Comon-Lundh, "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*, G. Ahn, M. Yung, and N. Li, Eds. ACM, 2014, pp. 609–620. [Online]. Available: https://doi.org/10.1145/2660267.2660276

[14] G. Scerri and R. Stanley-Oakes, "Analysis of key wrapping apis: Generic policies, computational security," in *CSF*. IEEE Computer Society, 2016, pp. 281–295.

[15] H. Comon and A. Koutsos, "Formal computational unlinkability proofs of RFID protocols," in *CSF*. IEEE Computer Society, 2017, pp. 100–114.

[16] G. Bana, R. Chadha, and A. K. Eeralla, "Formal analysis of vote privacy using computationally complete symbolic attacker," in *ESORICS (2)*, ser. Lecture Notes in Computer Science, vol. 11099. Springer, 2018, pp. 350–372.

[17] A. Koutsos, "The 5G-AKA authentication protocol privacy," in *EuroS&P*. IEEE, 2019, pp. 464–479.

[18] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, and S. Moreau, "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, 2021, pp. 537–554. [Online]. Available: https://doi.org/10.1109/SP40001.2021.00078

[19] D. Baelde, S. Delaune, A. Koutsos, and S. Moreau, "Cracking the stateful nut: Computational proofs of stateful security protocols using the squirrel proof assistant," in *CSF*. IEEE, 2022, pp. 289–304.

[20] D. Baelde, A. Koutsos, and J. Lallemand, "A Higher-Order Indistinguishability Logic for Cryptographic Reasoning," in *LICS'23*. ACM, 2023, to appear. [Online]. Available: https://inria.hal.science/hal-03981949

[21] M. Rusinowitch, R. Küsters, M. Turuani, and Y. Chevalier, "An np decision procedure for protocol insecurity with xor," in *Logic in Computer Science, Symposium on*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2003, p. 261. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/LICS.2003.1210066

[22] H. Comon-Lundh and V. Shmatikov, "Intruder deductions, constraint solving and insecurity decision in presence of exclusive or," in *LICS*. IEEE Computer Society, 2003, p. 271.

[23] S. Bursuc, H. Comon-Lundh, and S. Delaune, "Deducibility constraints and blind signatures," *Inf. Comput.*, vol. 238, pp. 106–127, 2014.

[24] H. Comon-Lundh, V. Cortier, and G. Scerri, "Tractable inference systems: An extension with a deducibility predicate," in *CADE*, ser. Lecture Notes in Computer Science, vol. 7898. Springer, 2013, pp. 91–108.

[25] The Squirrel Prover repository. https://github.com/squirrel-prover/squirrel-prover/.

[26] G. Barthe, T. Espitau, B. Grégoire, J. Hsu, L. Stefanesco, and P. Strub, "Relational reasoning via probabilistic coupling," in *LPAR*, ser. Lecture Notes in Computer Science, vol. 9450. Springer, 2015, pp. 387–401.

[27] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *ACM Trans. Comput. Syst.*, vol. 8, no. 1, p. 18–36, feb 1990. [Online]. Available: https://doi.org/10.1145/77648.77649

[28] G. Lowe, "Breaking and fixing the needham-schroeder public-key protocol using fdr," in *Tools and Algorithms for the Construction and Analysis of Systems*, T. Margaria and B. Steffen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 147–166.

[29] C. Cremers, C. Fontaine, and C. Jacomme, "A logic and an interactive prover for the computational post-quantum security of protocols," in *SP*. IEEE, 2022, pp. 125–141.

[30] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *POPL*. ACM, 1977, pp. 238–252.

[31] D. Monniaux, "Abstracting cryptographic protocols with tree automata," *Sci. Comput. Program.*, vol. 47, no. 2-3, pp. 177–202, 2003.

[32] M. Journault, A. Miné, and A. Ouadjaout, "An abstract domain for trees with numeric relations," in *ESOP*, ser. Lecture Notes in Computer Science, vol. 11423. Springer, 2019, pp. 724–751.

Appendices Outline: Appendix A details the modeling of a simple protocol, the Hash Lock protocol, in the indistinguishability logic. In Appendix B, we give missing details of the program semantics, and in Appendix C we develop the coupling method introduced in Section IV-C. In Appendix D, we present our complete proof system for bideduction. Finally,

Appendix E show how the Hash Lock protocol's security can be proven in SQUIRREL with the new tactics **crypto**, giving more details on how the **crypto** tactic is implemented.

APPENDIX A
PROTOCOLS MODELING

The indistinguishability logic of Section II forms the theoretical foundations of the SQUIRREL Prover, a proof assistant for the verification of security protocols. This protocol verification tool, introduced in [18], has been extended in various ways since then [19], [29].

We provide here a quick description of how protocols are modeled in SQUIRREL, through an example: the Hash Lock protocol. While understanding how protocol are modeled and analyzed in the logic of SQUIRREL is not necessary to understand the contributions of this article, it should help put this work in context. Essentially, a protocol is a distributed program that aims at providing some security properties. As usual, we consider an active adversary that fully controls the network: it can reads, intercept and even modify all messages exchanged by honest participants.

Example 13. *The Hash Lock protocol relies on on a keyed hash function* $\mathsf{h}(\_,\_)$*, and involves participants* $A_1, A_2, \ldots$ *where each* A_i *owns a secret hashing key* k_i *to be used across an unbounded number of sessions. For its* j^{th} *session, participant* A_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 would like that an adversary against the protocols, not knowing the keys* k_i*, but having obtained the outputs of several sessions on chosen inputs, cannot distinguish the next output from a random sampling. This is the case whenever* h *is assumed to be a PRF.*

We model an execution of a protocol along an execution trace with the adversary. Points of the execution trace, called timestamps, represent instants at which an interaction took place between the adversary and an honest participant, i.e. a participant inputted a message from the adversary and then outputted its answer. More precisely, we use the type timestamp to represent the execution trace, assumed to be always fixed, finite and well-founded w.r.t. $<$: timestamp $\to$ timestamp $\to$ bool. We assume special constants init, undef : timestamp representing, resp., the initial timestamp and a timestamp that never happens, and that the order $<$ is total on timestamps other than undef, and induces a predecessor function pred : timestamp $\to$ timestamp.

We model the execution of an adversary interacting with the protocol along a fixed execution trace using three mutually recursive functions input, output, frame : timestamp $\to$ message. We use a special syntax using @ to denote the application of these functions to timestamps: input@t represents the input provided by the adversary at time t; output@t the protocol output at that time; and frame@t the sequence of all outputs up to time t, included. The functions frame and input are always defined as follows:

$$\begin{aligned} \mathsf{frame@init} = \mathsf{input@init} &= \mathsf{output@init} = \mathsf{empty} \\ \mathsf{frame@\ t} &= \langle \mathsf{frame@pred\ t}, \mathsf{output@t}\rangle \\ \mathsf{input@\ t} &= \mathsf{att}(\mathsf{frame@\ pred\ t}) \qquad (\text{when } \mathsf{init} < \mathsf{t}) \end{aligned}$$

On the contrary, output depends on the protocol (it models the messages outputted by the participants). Once these functions are defined, they can be used to express the intended security properties of the protocol as a logical formula. We show on example of such a modeling for the Hash Lock protocol in the next example.

Example 14. *We use the types* timestamp *but also* index*, assumed to be tagged as fixed and finite and a function* $\mathsf{T} : \mathsf{index} \times \mathsf{index} \to \mathsf{timestamp}$ *so that* $\mathsf{T}(i,j)$ *represents a timestamp where tag* i *plays its session* j*. We view this function as a constructor that is injective, mutually exclusive, and exhaustive, through natural axioms.*

We model the execution of an adversary interacting with the Hash Lock protocol along a fixed execution trace using the three mutually recursive functions input*,* output*,* frame *presented before, and defining* output *as follows:*

$$\mathsf{output@t} = \mathsf{h}(\langle \mathsf{n}(i,j), \mathsf{input@t}\rangle, \mathsf{k}\ i) \qquad (\text{when } \mathsf{t} = \mathsf{T}(i,j))$$

Finally, the fact that the output at time t *is indistinguishable from a random value can then be expressed as follows, using a name* fresh : unit $\to$ message *which is not used in the protocol:*

$$\mathsf{frame@pred\ t}, \mathsf{output@t} \sim \mathsf{frame@pred\ t}, \mathsf{fresh}\ \langle\rangle$$

This equivalence can be proved in SQUIRREL *using (the tactic corresponding to) a PRF axiom scheme in the style of Example 2. As we shall see, we will prove it directly from the PRF game, by bi-deduction.*

APPENDIX B
PROGRAM SEMANTICS

We provide here the definitions of the semantics for our expressions and programs.

A. Expression Semantics

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. 5, 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:

$$[f]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu} \stackrel{\mathsf{def}}{=} [\![f]\!]^{\eta,(\mathfrak{p}|_{\mathrm{T_A},\mathsf{bool}},\rho_0)}_{\mathbb{M}:\mathcal{E}}$$

i.e. the (logical) semantics of f in the model $\mathbb{M}$, using $\mathfrak{p}|_{\mathrm{T_A},\mathsf{bool}}$ as adversarial (logical) random tape, and the all-zero random

$$[\mathsf{b}]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu} \stackrel{\text{def}}{=} i \qquad [v]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu} \stackrel{\text{def}}{=} \mu(v) \quad \text{when } v \in \mathcal{X}_{\mathsf{p}}$$
$$[f]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu} \stackrel{\text{def}}{=} [\![f]\!]^{\eta,(\mathfrak{p}|_{\mathtt{T_A},\mathsf{bool}},\rho_0)}_{\mathbb{M}:\mathcal{E}} \quad \text{when } f \in \mathcal{L}_{\mathsf{p}}$$
$$[e_1\ e_2]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu} \stackrel{\text{def}}{=} [e_1]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu}\,([e_2]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu})$$

Fig. 5. Semantics of expressions w.r.t. an model $\mathbb{M} : \mathcal{E}$.

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

Fig. 6. Initial memory of a game G w.r.t. $\mathbb{M}$ and side bit i.

tape ρ_0 as honest random tape — since all library function $\mathcal{L}_{\mathsf{p}}$ will be assumed to be adversarial, and therefore do not use *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. Initial Memory

The initial memory $\mu^{i\ \eta,\mathfrak{p}}_{\mathsf{init}\mathbb{M}}(\mathsf{G})$ 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. 6. It is obtained by evaluating the deterministic global variable assignments. Moreover, the value of the security parameter is made available to the game and adversary through the variable eta. Global random variables are not in this initial memory; they will be sampled during oracle calls.

C. 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** $(\mathsf{p};\ l \leftarrow \mathsf{tail}\ l)$ 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.

D. 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 $(\!|\mathsf{p}|\!)^{\eta,\mathfrak{p}}_{\mathsf{G},\mathbb{M},i,\mu} \in \mathsf{Mem}_{\mathbb{M},\eta} \cup \{\bot\}$ of a program p 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. Its definition, given in Fig. 7, 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 $v \xleftarrow{\$} \mathtt{T}[e]$ 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 $\mathfrak{p}|^{\eta}_{(\mathtt{T},\tau)}[k]$ from the random tape labeled by $(\mathtt{T}, \tau)$, 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., $f.\mathsf{args}$, $\mathsf{G}.\mathsf{glob}_{\$}$ and $f.\mathsf{loc}_{\$}$; then, we execute the oracle body $f.\mathsf{prog}$; and finally, we store the result of the evaluation of the return expression $f.\mathsf{expr}$ in v.

E. Adversaries

A adversary against G (or G-adversary) is a PTIME 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:

- We forbid the adversary from directly sampling from the $\mathtt{T}_{\mathsf{G}}$-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.

Appendix C
Probabilistic Couplings and Bi-Deduction

In this appendix we go back to Section IV-C, 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 15. *Consider a name* $\mathsf{n} : \mathsf{unit} \to \mathsf{bool}$ *and let* $\mathcal{C} = \{c_0, c_1\}$ *with:*

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

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

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

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

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

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

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

$$(\!|v \leftarrow O_f(\vec{e})[\vec{e_g};\vec{e_l}]|\!)^{\eta,\mathfrak{p}}_{\mu} \stackrel{\text{def}}{=} \text{let } \mu' = \mu \begin{bmatrix} f.\text{args} & \mapsto [\vec{e}]^{\eta,\mathfrak{p}}_{\mu} \\ \mathsf{G}.\text{glob}_{\$} & \mapsto \mathfrak{p}|^{\eta}_{\mathtt{T}_{\mathsf{G}}}[\,[\vec{e_g}]^{\eta,\mathfrak{p}}_{\mu}\,] \\ f.\text{loc}_{\$} & \mapsto \mathfrak{p}|^{\eta}_{\mathtt{T}_{\mathsf{G}}}[\,[\vec{e_l}]^{\eta,\mathfrak{p}}_{\mu}\,] \end{bmatrix} \text{ in}$$

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

$$\mu''\big[v \mapsto [f.\text{expr}]^{\eta,\mathfrak{p}}_{\mu''}\big]$$

Fig. 7. Program semantics w.r.t. a model $\mathbb{M} : \mathcal{E}$, a side $i \in \{0,1\}$ and a game G.

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 $\mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}$*. First observe that* $\rho\ \mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}\ \rho_a$ *imposes that* ρ_a *is a prefix of* $\mathfrak{p}[\mathtt{T}_{\mathtt{A}}, \mathsf{bool}]$ *and:*

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

Less formally, the logical tape must coïncide with the simulator tape on $\mathsf{n}\ \langle\rangle$ *when this sampling is zero; otherwise it must coïncide with the local sampling tape for that name. Thus, the program tape* ρ_a *such that* $\mathfrak{p}|^{\eta}_{\mathtt{T}_S}[O_{\mathbb{M},\eta}(\mathsf{n},\langle\rangle)] = 0$ *and* $\mathfrak{p}|^{\eta}_{\mathtt{T}^{\mathsf{loc}}_{\mathsf{G}}}[O_{\mathbb{M},\eta}(\mathsf{n},\langle\rangle)] = 1$ *is not related to any logical tape in* $\mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}$ *— for any* ρ*, we do not have* $\rho\ \mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}\ \rho_a$*. Hence the right marginal of a coupling included in* $\mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}$ *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.*

A. Preliminaries: Probability Theory

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

1) 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 $\forall E \in \mathcal{F}_\mathbb{S},\ X^{-1}(E) \in \mathcal{F}_\Omega$.

2) 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 $\Pr(X \in E) \stackrel{\text{def}}{=} \mu(X^{-1}(E))$.

3) Distributions as programs: We will describe some distributions using programs written in pseudo-code, e.g. if D is a distribution, then the program $\mathsf{x} \xleftarrow{\$} D; \mathsf{y} \xleftarrow{\$} D; \textbf{return } (\mathsf{x},\mathsf{x}+\mathsf{y})$ 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.

4) 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 $\mathcal{F} = L$, from which we deduce that D_1 and D_2 coincides on L. □

B. Couplings and Lifting Lemma

Recall that, in section Section IV-C, in order to be able to *lift* equalities over tapes in $\mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}$ to equalities over probabilities, we relied on the standard notion of a probabilistic coupling and lifting (as in [26]). TIn 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 $\mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}$, 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:*

$$\text{for all } x\ R\ y,\ \ x \in E_1 \text{ iff. } y \in E_2. \tag{4}$$

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. (4):

$$(E_1 \times \mathbb{S}_2) \cap R = (E_1 \times E_2) \cap R \tag{5}$$

and

$$(\mathbb{S}_2 \times E_2) \cap R = (E_1 \times E_2) \cap R. \tag{6}$$

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

$$\begin{aligned}
\Pr_{\mu_1}(E_1) &= \Pr_{\mu}(E_1 \times \mathbb{S}_2) && \text{(left marginal property)}\\
&= \Pr_{\mu}((E_1 \times \mathbb{S}_2) \cap R) && \text{(by containement)}\\
&= \Pr_{\mu}((E_1 \times E_2) \cap R) && \text{(by Eq. (5))}\\
&= \Pr_{\mu}((\mathbb{S}_1 \times E_2) \cap R) && \text{(by Eq. (6))}\\
&= \Pr_{\mu}(\mathbb{S}_1 \times E_2) && \text{(by containement)}\\
&= \Pr_{\mu_2}(E_2) && \text{(right marginal property)}
\end{aligned}$$

which concludes this proof. □

C. Well-formedness of Constraints System

Before moving on to the proof of Lemma 1, we are missing one key ingredient: the notion of well-formedness of a constraints system. First, we defined the restriction of a logical random tape to the (concrete) names constrained by a constraint system.

Definition 11 (Restriction of a random tapes)**.** *Let $\mathbb{M} : \mathcal{E}$ be a model, η a security parameter, ρ be a random tape and $\mathcal{C}$ a constraint system. The restriction of ρ by $\mathcal{C}$ with w.r.t. $\mathbb{M}$ and η — written $\rho_{|\mathbb{M},\eta,\mathcal{C}}$ — is the random tape obtained from ρ by zeroing all random bits that corresponds to names that are **not** in $\mathcal{N}^{\eta,\rho}_{\mathcal{C},\mathbb{M}}$.*

We define well-formedness as follows.

Definition 12 (Well-formedness of constraint systems)**.** *Let $\mathbb{M} : \mathcal{E}$ be a model and η the security parameter. A constraint system $\mathcal{C}$ is* well-formed with respect to $\mathbb{M}, \eta$ *when:*

- *$\mathcal{C}$ is empty;*
- *or $\mathcal{C} = \mathcal{C}_0, (\vec{\alpha}, n, t, T, f)$ where $\mathcal{C}_0$ is a well-formed constraint system w.r.t. $\mathbb{M}, \eta$ and there exists a (mathematical and deterministic) function g such that for all random tape ρ and valuation $\vec{a} \in [\![\vec{\tau}]\!]^{\eta}_{\mathbb{M}}$ (where $\vec{\tau}$ are the types of $\vec{\alpha}$),*

$$g(\rho_{|\mathbb{M}[\vec{\alpha} \mapsto \vec{a}],\eta,\mathcal{C}_0}) = [\![(t \mid f)]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha} \mapsto \vec{a}]:\mathcal{E}}.$$

We write $\mathcal{E}, \Theta \models_{\mathsf{WF}} \mathcal{C}$ if $\mathcal{C}$ is well-formed w.r.t. $\mathbb{M}, \eta$ for any $\mathbb{M} : \mathcal{E}$ such that $\mathcal{E}, \Theta \models \mathbb{M}$. For pairs $\mathcal{C}_{\#} = \#(\mathcal{C}_0, \mathcal{C}_1,)$ of constraint systems, $\mathcal{E}, \Theta \models_{\mathsf{WF}} \mathcal{C}_{\#}$ stands for well-formedness of both $\mathcal{C}_0$ and $\mathcal{C}_1$.

Outline: The following two sections of this appendix aims at proving Lemma 1, i.e. that a probabilistic coupling contained in $\mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}$ can be constructed from any well-formed and valid constraint systems $\mathcal{C}$. 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-D, and we then use this result to prove Lemma 1 in Appendix C-E.

D. Couplings Arrays

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

1) 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 13. *We let $D_{\mathbb{S}}^{\mathbb{I}}$ 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):*

$$\begin{array}{l} \mathsf{a} \leftarrow [\bot \textit{ for } \_ \in \mathbb{I}]; \\ \textit{for } (j \in \mathbb{I}) \textit{ do } \{\, \mathsf{a}[j] \xleftarrow{\$} D_{\mathbb{S}}; \,\} \\ \textbf{return } \mathsf{a}; \end{array} \tag{7}$$

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:*

$$\begin{array}{l} \mathsf{a} \leftarrow [\bot \text{ for } \_ \in \mathbb{I}]; \\ \mathsf{s} \leftarrow \mathsf{s}_{\mathsf{init}}; \\ \textbf{for } (\_ \in |\mathbb{I}|) \textbf{ do } \{\, i \leftarrow \mathsf{f}(\mathsf{s}); \, \mathsf{a}[i] \xleftarrow{\$} D_{\mathbb{S}}; \, \mathsf{s} \leftarrow \mathsf{g}(\mathsf{s}, \mathsf{a}); \,\} \\ \textbf{return } \mathsf{a}; \end{array} \tag{8}$$

where $\mathsf{s}_{\mathsf{init}}$, 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:

$$\Pr_p(\mathsf{a} \in \textstyle\prod_i E_i) = \Pr_{p_0}(\mathsf{a} \in \prod_i E_i) \tag{9}$$

where p_0 is the program sampling the array in an i.i.d. fashion as described in Eq. (7) (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. (9).

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. (9)). 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 $D_{\mathbb{S}}^{\mathbb{I}}$. □

2) 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 $a_1 : \mathbb{I}_1 \to \mathbb{S} \cup \{\bot\}$ and $a_2 : \mathbb{I}_2 \to \mathbb{S} \cup \{\bot\}$, $(\mathsf{select}\ a_1\ a_2)$ either select a pair of $\bot$-valued indices of a_1 and a_2, or return a special value done. More precisely:

$$\begin{aligned} & \forall a_1, a_2.\ \mathsf{select}\ a_1\ a_2 \in (\mathbb{I}_1 \times \mathbb{I}_2) \cup \{\mathsf{done}\} \\ & \text{and } \mathsf{select}\ a_1\ a_2 = (i_1, i_2) \Rightarrow a_1[i_1] = \bot \wedge a_2[i_2] = \bot \end{aligned} \tag{10}$$

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. (10), 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. □

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

We now recall and prove Lemma 1.

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

Proof. Let $\mathbb{M}$ be a model and $\mathcal{C}$ a constraint system such that $\mathcal{C}$ 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 $\mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}$.

Given $\mathcal{C}$, $\mathbb{M}$ and η, 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 $(\mathsf{select}\ \rho\ \mathfrak{p})$ with which we instantiate the framework. Our selection function looks for an elementary constraint $c = (\vec{\alpha}, \mathsf{n}, t, T, f) \in \mathcal{C}$ and a value $\vec{a} \in [\![\vec{\tau}]\!]^{\eta}_{\mathbb{M}}$ (where $\vec{\tau}$ are the types of $\vec{\alpha}$) 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.

If such c and $\vec{a}$ exist, choose an arbitrary one and return the corresponding indices in the logical and computational tapes. Otherwise, return done.

Our selection function, making a choice among an infinite domain $[\![\vec{\tau}]\!]^{\eta}_{\mathbb{M}}$, is ineffective. Note, however, that $p_c(\mathsf{select})$ still only takes a finite number of iterations, as each iteration removes an undefined position in tape ρ, which is finite. More precisely, it is useful to declare two pairs $(c, \vec{a})$ and $(c', \vec{a'})$ as equivalent when 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, we note that the effect of selecting a pair $(c, \vec{a})$ is the same as selecting any equivalent $(c', \vec{a'})$. This is interesting because, even though there might be infinitely many possible values for the $\vec{\alpha}$ variables, the indices t can only take a finite number of values (recall that $\mathbb{M}$ and η are fixed).

We now show that our coupling is contained in $\mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}$. To do so, consider an arbitrary run of $p_c(\mathsf{select})$. We say that a pair $(c, \vec{a})$ is addressed at some point in this run if satisfies (a) but neither (b) nor (c). Once a pair is addressed, the value of the corresponding name will have been set in the tapes. Note, though that a pair needs not be selected to be addressed: it suffices that an equivalent pair is selected. First, we observe that, at every step of our run, and for every pair $(c, \vec{a})$ 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. Second, we note that, for every pair $(c, \vec{a})$, condition (a) will eventually be met. This is a consequence of well-foundedness: assume for the sake of contradiction that this is not the case for a constraint c such that $\mathcal{C} = \mathcal{C}_0 \cdot c \cdot \mathcal{C}_1$, and consider the leftmost such c. At some point, all pairs $(c', \vec{b})$ with $c' \in \mathcal{C}_0$ will satisfy (a) and, after a finite number of iterations, they will all have been addressed. From then on, condition (a) will hold for $(c, \vec{a})$. To conclude, every pair will eventually satisfy (a) and must thus eventually be addressed. Hence, for any $(n, v, \mathtt{T}) \in \mathcal{N}^{\eta,\rho}_{\mathcal{C},\mathbb{M}}$, we have $[\![n]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(v) = \mathfrak{p}|^{\eta}_{\mathtt{T}}[O_{\mathbb{M},\eta}(n, v)]$. The rest of $\mathcal{R}^{\eta}_{\mathcal{C},\mathbb{M}}$, concerning ρ_a and $\mathfrak{p}[\mathtt{T}_S, \mathsf{bool}]$ is obvious. □

Appendix D
Proof System

In this section we present the full proof system, partially given in the article.

A. 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. This simulator must be an adversary, and, as such, be a polytime program. Besides, to bideduce some terms, one might need a simulator with a while-loop, e.g. to compute a function graph. To ensure such simualator 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.

Futhermore, $\mathsf{well\text{-}founded}_{\tau}(<)$ 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.

B. Inference rules

Inference rules of our proof system, given in Fig. 8, Fig. 9, and Fig. 10, are organized in three categories:

- First, the *constructive* rules. This includes weakening rules (of hypotheses, pre- and post-conditions, constraints $\cdots$), 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 adversary. This comprises function applications, transitivity, computation of adversarial terms,

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

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

$$\textsc{Weak.Mem}\ \frac{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi',\psi')\vdash \vec{u_\#}\rhd\vec{v_\#} \quad \mathcal{E},\Theta\models^A\varphi\Rightarrow\varphi' \quad \mathcal{E},\Theta\models^A\psi'\Rightarrow\psi}{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi,\psi)\vdash \vec{u_\#}\rhd\vec{v_\#}}$$

$$\textsc{Weak.Hyps}\ \frac{\mathcal{E},\Theta',\mathcal{C}_\#,(\varphi_\#,\psi_\#)\vdash \vec{u_\#}\rhd\vec{w_\#} \quad \Theta\models\Theta'}{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#)\vdash \vec{u_\#}\rhd\vec{w_\#}}$$

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

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

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

$$\textsc{Drop}\ \frac{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#)\vdash \vec{u_\#}\rhd\vec{v_\#},\vec{t_\#}}{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#)\vdash \vec{u_\#}\rhd\vec{v_\#}}$$

$$\textsc{Rewrite-L}\ \frac{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#)\vdash \#(\vec{w_0},\vec{w_1})\rhd v_\# \quad \mathcal{E},\Theta\vdash[\vec{u_0}=\vec{w_0}]_{\mathrm{e}}\tilde{\wedge}[\vec{u_1}=\vec{w_1}]_{\mathrm{e}}}{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#)\vdash \#(\vec{u_0},\vec{u_1})\rhd v_\#}$$

$$\textsc{Rewrite-R}\ \frac{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#)\vdash \vec{u_\#}\rhd\#(\vec{w_0},\vec{w_1}) \quad \mathcal{E},\Theta\vdash[\vec{v_0}=\vec{w_1}]_{\mathrm{e}}\tilde{\wedge}[\vec{v_0}=\vec{w_1}]_{\mathrm{e}}}{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#)\vdash \vec{u_\#}\rhd\#(\vec{v_0},\vec{v_1})}$$

$$\textsc{Dup}\ \frac{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#)\vdash \vec{u_\#}\rhd\vec{v_\#},t_\#}{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#)\vdash \vec{u_\#}\rhd\vec{v_\#},t_\#,t_\#}$$

Fig. 8. Bi-deduction constructive rules

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

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

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

$$\vec{o}_\#=(o_{v\sharp})_{v\in f.\mathrm{glob}_\$}\ \text{ and }\ \vec{s}_\#=(s_{v\sharp})_{v\in f.\mathrm{glob}_\$}$$

Fig. 9. Bi-deduction adversarial rules

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 notice that, for all $\mathbb{M}$ and η, we have:

- $\mathrm{Valid}(\mathcal{C}^1\cdot\mathcal{C}^2)\models\mathrm{Valid}(\mathcal{C}^1)\wedge\mathrm{Valid}(\mathcal{C}^2)$,
- for all $j\in\{1,2\}$, $\mathcal{R}^\eta_{\mathcal{C}^1\cdot\mathcal{C}^2,\mathbb{M}}\subseteq\mathcal{R}^\eta_{\mathcal{C}^j,\mathbb{M}}$, and
- for all $j\in\{1,2\}$ and random tape $\rho:\mathcal{N}^{\eta,\rho}_{\mathcal{C}^j,\mathbb{M}}\subseteq\mathcal{N}^{\eta,\rho}_{\mathcal{C}^1\cdot\mathcal{C}^2,\mathbb{M}}$
- For two bi-constraints systems $\mathcal{C}_\#$ and $\mathcal{C}'_\#$, if $\mathcal{E},\Theta\models_{\mathsf{WF}}\mathcal{C}_\#$ and $\mathcal{E},\Theta\models_{\mathsf{WF}}\mathcal{C}'_\#$ then $\mathcal{E},\Theta\models_{\mathsf{WF}}(\mathcal{C}_\#\cdot\mathcal{C}'_\#)$.

We have similar properties for constraint generalization:

- $\mathrm{Valid}(\forall(x:\tau).\mathcal{C})\models\tilde{\forall}(x:\tau).\mathrm{Valid}(\mathcal{C})$
- for all $a\in[\![\tau]\!]^\eta_{\mathbb{M}}$, $\mathcal{R}^\eta_{\forall(x:\tau).\mathcal{C},\mathbb{M}}\subseteq\mathcal{R}^\eta_{\mathcal{C},\mathbb{M}[x\mapsto a]}$
- for all $a\in[\![\tau]\!]^\eta_{\mathbb{M}}$ and random tape ρ: $\mathcal{N}^{\eta,\rho}_{\mathcal{C},\mathbb{M}[x\mapsto a]}\subseteq\mathcal{N}^{\eta,\rho}_{\forall(x:\tau).\mathcal{C},\mathbb{M}}$
- For any bi-constraints system $\mathcal{C}$ and variable $x:\tau$, if $\mathcal{E},x:\tau,\Theta\models_{\mathsf{WF}}\mathcal{C}_\#$ then $\mathcal{E},\Theta\models_{\mathsf{WF}}\forall(x:\tau).\mathcal{C}_\#$.

1) Constructive Rules: The common point of all constructive rules is that they don't change the adversary given by the premise.

First, notice that an adversary 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 doesn't change the adversary results. 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

$$\textsc{Adversarial}\ \frac{\mathcal{E},\Theta \vdash \mathsf{adv}(t)}{\mathcal{E},\Theta,\emptyset,(\varphi_\#,\varphi_\#) \vdash \vec{u_\#} \rhd t}$$

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

$$\textsc{If-then-else}\ \frac{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#) \vdash \vec{u_\#} \rhd \vec{v_\#},(b_\# \mid f_\#),(t_\# \mid f_\# \wedge b_\#),(t'_\# \mid f_\# \wedge \neg b_\#)}{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#) \vdash \vec{u_\#} \rhd \vec{v_\#},(\mathsf{if}\ b_\#\ \mathsf{then}\ t_\#\ \mathsf{else}\ t'_\# \mid f_\#)}$$

$$\textsc{Transitivity}\ \frac{\mathcal{E},\Theta,\mathcal{C}^1_\#,(\varphi_\#,\varphi'_\#) \vdash \vec{u_\#} \rhd \vec{t_\#} \quad \mathcal{E},\Theta,\mathcal{C}^2_\#,(\varphi'_\#,\psi_\#) \vdash \vec{u_\#},\vec{t_\#} \rhd \vec{v_\#}}{\mathcal{E},\Theta,\mathcal{C}^1_\# \cdot \mathcal{C}^2_\#,(\varphi_\#,\psi_\#) \vdash \vec{u_\#} \rhd \vec{t_\#},\vec{v_\#}}$$

$$\textsc{Lambda App}\ \frac{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#) \vdash \vec{u_\#} \rhd (g_\#,t_\# \mid f_\#) \quad \mathcal{E} \vdash \vec{t_\#} : \tau \qquad \mathsf{enum}(\tau)}{\mathcal{E},\Theta,\mathcal{C}_\#,(\varphi_\#,\psi_\#) \vdash \vec{u_\#} \rhd (g_\#\ t_\# \mid f_\#)}$$

$$\textsc{Lambda}\ \frac{(\mathcal{E},x:\tau),\Theta,\mathcal{C}_\#,(\varphi_\#,\varphi_\#) \vdash \vec{u_\#} \rhd (t_\# \mid f_\#) \quad \mathcal{E},x:\tau \vdash t_\# : \tau_\mathsf{b} \qquad \mathsf{enum}(\tau)}{\mathcal{E},\Theta,\forall(x:\tau).\mathcal{C}_\#,(\varphi_\#,\varphi_\#) \vdash \vec{u_\#} \rhd (\lambda(x:\tau).t_\# \mid f_\#)}$$

$$\textsc{Quantificator}\ O \in \{\forall,\exists\}\ \frac{(\mathcal{E},x:\tau),\Theta\ \tilde{\wedge}\ \mathsf{adv}(x),\mathcal{C}_\#,(\varphi_\#,\varphi_\#) \vdash \vec{u_\#} \rhd (t_\# \mid f_\#) \qquad \mathsf{enum}(\tau)}{\mathcal{E},\Theta,\forall(x:\tau).\mathcal{C}_\#,(\varphi_\#,\varphi_\#) \vdash \vec{u_\#} \rhd (O(x:\tau).t_\# \mid f_\#)}$$

$$\textsc{Induction}\ \frac{(\mathcal{E},x:\tau),\Theta,\mathcal{C}_\#,(\varphi_\#,\varphi_\#) \vdash \vec{u_\#},(\lambda(y:\tau).\mathsf{if}\ y<x\ \mathsf{then}\ t[x\mapsto y] \mid f_\#) \rhd (t_\# \mid f_\#) \quad \mathsf{finite}(\tau) \quad \mathsf{fixed}(\tau) \quad \mathcal{E},\Theta \vdash \mathsf{well\text{-}founded}_\tau(<)\ \tilde{\wedge}\ \mathsf{adv}(<)}{\mathcal{E},\Theta,\forall(x:\tau).\mathcal{C}_\#,(\varphi_\#,\varphi_\#) \vdash \vec{u_\#} \rhd (\lambda(x:\tau).\,t_\# \mid f_\#)}$$

Fig. 10. Bi-deduction computational rules

previous remark that when $\mathcal{C} \subseteq \mathcal{C}'$ then $\mathrm{Valid}(\mathcal{C}') \Rightarrow \mathrm{Valid}(\mathcal{C})$. Finally the rule Weak.Cond weaken the local formula attached to term. For any term $(t_\# \mid f_\#)$ and a formula $f'_\#$ such that $[f' \Rightarrow f_\#]_\mathrm{e}$ 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.

2) Computational Rules: Computational rules includes rules that do not require random samplings or oracle calls from the adversary:

- the rule Adversarial to build a program that compute an adversary 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 an 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.

3) 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 14 (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 $\vec{k_\#} = (k_v\ o_{v\sharp})_{v\in f.\mathrm{glob_\$}}$ and $\vec{r_\#} = (r_v\ s_{v\sharp})_{v\in f.\mathrm{loc_\$}}$.

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:

$$[\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = \mu'_i(X)[f.\mathrm{expr}]^{\eta,\mathfrak{p}}_{\mathbb{M}:\mathcal{E},i,\mu'_i}$$
$$\text{where } \mu'_i = (\!|X \leftarrow O_f([\![\vec{t_i}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})[\vec{e_k};\vec{e_s}]|\!)^{\eta,\mathfrak{p}}_{\mu_i}$$
$$\vec{e_k} = \left(O_{\mathbb{M}:\mathcal{E},\eta}(k_v,[\![o_{v,i}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})\right)_{v\in f.\mathrm{glob_\$}}$$
$$\vec{e_s} = \left(O_{\mathbb{M}:\mathcal{E},\eta}(r_v,[\![s_{v,i}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})\right)_{v\in f.\mathrm{loc_\$}}$$

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:*

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

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

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 $\mathcal{C}$ 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 form the validity of the Hoare triplet. □

Example 16. *Let's take the PRF game of example Example 1 and show the indistinguishability*

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

using bi-deduction with the PRF game, with $\mathsf{n},\mathsf{k},\mathsf{m},\mathsf{s}$ *distinct names of type* τ*.*

The plan is to show that the first and last terms can be computed by oracle calls and the middle one is just function applications. For that we will need that h *is adversarial, but also a way to ensure that* $\mathsf{n} \neq \mathsf{m}$*. For this latter point would like to use the tricks of example 11. Since* $\mathsf{large}(\tau) \models \mathsf{n} \neq \mathsf{m} \sim true$*, the formulas*

$$\begin{aligned} &\mathsf{h}(\mathsf{n},\mathsf{k}), \mathsf{h}(\mathsf{m},\mathsf{s}), \mathsf{if}\ \mathsf{n} \neq \mathsf{m}\ \mathsf{then}\ \mathsf{h}(\mathsf{m},\mathsf{k}) \\ \sim\ &\mathsf{h}(\mathsf{n},\mathsf{k}), \mathsf{h}(\mathsf{m},\mathsf{s}), \mathsf{if}\ \mathsf{n} \neq \mathsf{m}\ \mathsf{then}\ \mathsf{fresh} \end{aligned}$$

implies our goal formula under $\mathsf{large}(\tau)$ *hypothesis (this is formula rewriting, see [18] for details). Let's take hypothesis* $\Theta = \mathsf{adv}(\mathsf{h}) \wedge \mathsf{large}(\tau)$*.*

For the sake of simplicity, we instantiate assertion by memory sets: the satisfiability then become inclusion and the implication inclusion.

Let

$$\varphi_0 = \{[l_{\mathsf{hash}} \mapsto []; l_{\mathsf{chal}} \mapsto []]$$

and

$$\varphi = \{[l_{\mathsf{hash}} \mapsto [n]; l_{\mathsf{chal}} \mapsto []]\}.$$

We have

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

and, using NAME*, gets the bi-deduction judgment*

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

Then, using NAME*,* DUP *and* FA*, we also gets that:*

$$\mathcal{E}, \Theta, ((\emptyset, \mathsf{n}, \emptyset, \mathtt{T}_S, \top), (\emptyset, \mathsf{m}, \emptyset, \mathtt{T}_S, \top), (\emptyset, \mathsf{s}, \emptyset, \mathtt{T}_S, \top), (\varphi, \varphi) \vdash \emptyset \rhd \mathsf{h}(\mathsf{m},\mathsf{s}), \mathsf{n} \neq \mathsf{m}$$

Finally, we have:

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

for a certain ψ*. As before, using the* IF-THEN-ELSE*,* $\textsc{Oracle}_f$ *for* $f = \mathsf{chal}$ *and* NAME *rules, we get*

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

By transitivity, we get the final judgement:

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

where:

$$\begin{aligned} \mathcal{C} = \{&(\emptyset, \mathsf{n}, \emptyset, \mathtt{T}_S, \top), (\emptyset, \mathsf{k}, \emptyset, \mathtt{T}^{\mathsf{glob}}_{\mathsf{G},k}, \top) \\ &(\emptyset, \mathsf{m}, \emptyset, \mathtt{T}_S), (\emptyset, \mathsf{s}, \emptyset, \mathtt{T}_S), \\ &(\emptyset, \mathsf{n}, \emptyset, \mathtt{T}_S, \top), (\emptyset, \mathsf{m}, \emptyset, \mathtt{T}_S, \top), \\ &(\emptyset, \mathsf{m}, \emptyset, \mathtt{T}_S, \mathsf{n} \neq \mathsf{m}), \\ &(\emptyset, \mathsf{k}, \emptyset, \mathtt{T}^{\mathsf{glob}}_{\mathsf{G},k}, \mathsf{n} \neq \mathsf{m}), \\ &(\emptyset, \mathsf{fresh}, \emptyset, \mathtt{T}^{\mathsf{loc}}_{\mathsf{G}}, \mathsf{n} \neq \mathsf{m})\} \end{aligned}$$

Then $\mathrm{Valid}(\mathcal{C}) = \top$*, and the proof is done.*

C. 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* $\mathbb{M}, \eta, \rho, \mu^{i}_{init\mathbb{M}}{}^{\eta,\rho}(\mathsf{G}) \models^{A} \varphi_i$*. The following rule is sound w.r.t. models where* G *is secure, for any* $\mathcal{C}_\#$*,* $\vec{v_\#}$ *and* $\psi_\#$*:*

$$\textsc{Bi-deduce}\quad \frac{\mathcal{E}, \Theta \vdash \mathrm{Valid}(\mathcal{C}_\#) \qquad \mathcal{E}, \Theta, \mathcal{C}_\#, (\varphi_\#, \psi_\#) \vdash \emptyset \rhd_{\mathsf{G}} \vec{v_\#}}{\mathcal{E}, \Theta \vdash \vec{v_0} \sim \vec{v_1}}$$

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:

$$\eta \mapsto \left| Pr_{\rho \in \mathbb{T}_{\mathbb{M},\eta}}\left(\mathcal{D}([\![\vec{v_0}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, 1^\eta, \rho_{\mathsf{a}}) = 1\right) - Pr_{\rho \in \mathbb{T}_{\mathbb{M},\eta}}\left(\mathcal{D}([\![\vec{v_1}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, 1^\eta, \rho_{\mathsf{a}}) = 1\right) \right| \tag{11}$$

Assume further that the premises are valid. Since $\mathcal{C}_\#$ is valid, we know that $\mathcal{C}_0$ and $\mathcal{C}_1$ are both valid. By the second premise, and by hypothesis on $\varphi_\#$, there exists a 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 $\rho\, \mathcal{R}^{\eta}_{\mathcal{C}_i,\mathbb{M}}\, \mathfrak{p}$, where μ_i is $\mu^{i}_{\mathsf{init}\mathbb{M}}{}^{\eta,\rho}$.

We now construct a 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:

$$\text{for all } \eta,\ i,\ \mu,\ \vec{a} \in [\![\vec{\tau}]\!]^{\eta}_{\mathbb{M}},\ \text{for all tapes } (\rho_{\mathsf{a}}, \rho_{\mathsf{h}})\ \mathcal{R}^{\eta}_{\mathcal{C}_i,\mathbb{M}}\ \mathfrak{p},\quad (\!|\mathsf{d}|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu[\vec{X}\mapsto\vec{a}]}[\mathsf{res}] = \mathcal{D}(\vec{a}, 1^{\eta}, \rho_{\mathsf{a}}) \quad (12)$$

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 $\mathfrak{p}[\mathtt{T}_{\mathbb{A}}, \mathsf{bool}]$, hence it is an adversary. We can thus form an adversary q by composing d with p. $\mathcal{C}_0$ is well-formed, hence, by Lemma 1, there exists a coupling $\mathbb{C} : \mathbb{T}_{\mathbb{M},\eta} \bowtie \mathfrak{P}$ contained in $\mathcal{R}^{\eta}_{\mathcal{C}_i,\mathbb{M}}$.

Hence using Eq. (12) and Lemma 2, we obtain that:

$$Pr_{\mathfrak{p}}\left((\!|\mathsf{q}|\!)^{\eta,\mathfrak{p}}_{\mathbb{M},0,\mu_0}[\mathsf{res}] = 1\right) = Pr_{\rho\in\mathbb{T}_{\mathbb{M},\eta}}\left(\mathcal{D}([\![\vec{v_0}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, 1^{\eta}, \rho_{\mathsf{a}}) = 1\right).$$

Repeating the reasoning for $i = 1$, we obtain that the following function is equal to the one in Eq. (11):

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

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

Appendix E
Implementation

We detail here the fully automated procedure search$_{\triangleright}$ for finding bi-deduction proofs using the proof system of Section IV. As explained in section Section V, we aim at a fully automated procedure, reasonnably 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.

A. Recursive Functions

Our logic supports recursively defined functions, which are crucially used to model protocols, as shown in Example 14. 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 $x_{\#} \leq u_{\#}$. 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 VI. When trying to verify a bi-deduction judgement $\vec{u_{\#}} \triangleright \vec{v_{\#}}$, we first search for induction-free derivations of judgements of the following form (we will discuss later how these judgments are determined):

$$(\mathcal{E}, t : \mathsf{timestamp}), \Theta, \mathcal{C}^i_{\#}, (\varphi_{\#}, \varphi_{\#}) \vdash \vec{u_{\#}}, (\lambda t'.\ \mathsf{if}\ t' < t \wedge t \leq t_0\ \mathsf{then}\ \vec{w_{\#}}[t \mapsto t']) \triangleright (w^i_{\#} \mid t \leq t_0) \quad (13)$$

$$\mathcal{E}, \Theta, \mathcal{C}'_{\#}, (\varphi_{\#}, \psi_{\#}) \vdash \vec{u_{\#}}, (\lambda t.\ \mathsf{if}\ t \leq t_0\ \mathsf{then}\ \vec{w_{\#}}) \triangleright \vec{v_{\#}} \quad (14)$$

The rough idea is that, instead of bi-deducing $\vec{u_{\#}} \triangleright \vec{v_{\#}}$, we more generally try to bi-deduce:

$$\vec{u_{\#}} \ \triangleright (\lambda t.\ \mathsf{if}\ t \leq t_0\ \mathsf{then}\ \vec{w_{\#}}), \vec{v_{\#}}$$

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:

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

using rules Transitivity, Induction and Drop and the derivations of the above judgements, with a derivation of Eq. (13) for each $i \in [1; k]$. Note that we restrict to have the same pre- and post-condition φ on all judgments of Eq. (13). 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 [19], which balances efficiency and precision.

Example 17. *In Example 14 we seek to prove the following indistinguishability, where* fresh *is a name which is not used in the protocol:*

$$\Theta \models \mathit{frame@pred}\ t_0, \mathsf{h}(\langle \mathsf{n}\ (i_0, j_0), \mathsf{att}(\mathit{frame@pred}\ t_0)\rangle, \mathsf{k}\ i_0) \sim \mathit{frame@pred}\ t_0, \mathsf{fresh}\ \langle\rangle$$

For the sake of simplicity, we shall assume here that $t_0 = T(i_0, j_0)$ for some indices that 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 $\mathcal{C}$ that is valid and any pre-condition φ that holds on the game's initial memory:

$$\mathcal{E}, \Theta, \mathcal{C}, (\varphi, \psi) \vdash \emptyset \triangleright \mathit{frame@pred}\ t_0, \mathsf{if}\ f_{\mathrm{Fresh}}\ \mathsf{then}\ \#(\mathsf{h}(\langle \mathsf{n}\ (i_0, j_0), \mathsf{att}(\mathit{frame@pred}\ t_0)\rangle, \mathsf{k}\ i_0), \mathsf{fresh}\ \langle\rangle)$$

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

We choose $\vec{w_{\#}} = (\mathit{frame@t}, \mathit{input@t}, \mathit{output@t})$ to prove this judgment using the strategy defined above, with $t :=$

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

pred t_0. *We will thus have to prove the following judgments, for some constraint systems* $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}'$ *and* $w'_\# = (\lambda t'.\ \mathsf{if}\ t' < t\ \mathsf{then}\ (\mathit{frame@}t', \mathit{input@}t', \mathit{output@}t') \mid t < t_0)$*:*

$$\mathcal{E}, \Theta, \mathcal{C}_1, (\varphi, \varphi) \vdash w'_\# \rhd (\mathit{frame@}t \mid t < t_0)$$
$$\mathcal{E}, \Theta, \mathcal{C}_2, (\varphi, \varphi) \vdash w'_\# \rhd (\mathit{input@}t \mid t < t_0)$$
$$\mathcal{E}, \Theta, \mathcal{C}_3, (\varphi, \varphi) \vdash w'_\# \rhd (\mathit{output@}t \mid t < t_0)$$
$$\mathcal{E}, \Theta, \mathcal{C}', (\varphi, \varphi) \vdash (\lambda t.\ \mathsf{if}\ t < t_0\ \mathsf{then}\ \vec{w_\#}) \rhd \mathit{frame@pred}\ t_0, \mathsf{if}\ f_{\mathrm{Fresh}}\ \mathsf{then}\ \#(\mathit{output@}t_0, \mathsf{fresh}\ \langle\rangle)$$

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 20.

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.

B. 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 $\vec{\alpha}$:

$$[\![(\vec{\alpha}, m, f)]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} \stackrel{\mathsf{def}}{=} \{[\![m]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha}\mapsto\vec{a}]:\mathcal{E}} \mid \vec{a} \in [\![\vec{\tau}]\!]^{\eta}_{\mathbb{M}}, [\![f]\!]^{\eta,\rho}_{\mathbb{M}[\vec{\alpha}\mapsto\vec{a}]:\mathcal{E}} = 1\}$$

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:

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

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 18. *To obtain a complete derivation in Example 17, we can use the following assertions:*

$$\varphi(\ell_{\mathsf{hash}}) = \{(\{i,j\}, \mathsf{n}\ (i,j), \mathcal{T}(i,j) < t_0)\} \qquad \psi(\ell_{\mathsf{hash}}) = \varphi(\ell_{\mathsf{hash}})$$
$$\varphi(\ell_{\mathsf{chal}}) = \emptyset \qquad \psi(\ell_{\mathsf{chal}}) = \{(\emptyset, \mathsf{n}\ (i_0, j_0), \mathsf{true})\}$$

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* $(\{j\}, \mathsf{n}, (i_0, j), \mathcal{T}(i_0, j) < t_0)$*. 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.*

C. Proof Search

In order to find the induction-free derivations required in the general proof method described in Appendix E-A, we implement the goal-directed heuristic proof search procedure $\mathrm{search}_\rhd(\cdot)$ 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, constraints $\mathcal{C}_\#$ such that $\mathcal{E}, \Theta \models_{\mathsf{WF}} \mathcal{C}_\#$, pre-condition $\varphi_\#$, input terms $\vec{u_\#}$, and output terms $\vec{v_\#}$,

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

In case the proof search fails, $\mathrm{search}_\rhd(\cdot)$ 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 17, 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 $\mathcal{C}_\#$ 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 $\mathsf{if}\ i = j\ \mathsf{then}\ \mathsf{h}(m, \mathsf{k}\ i)\ \mathsf{else}\ \mathsf{h}(m, \mathsf{k}\ i)$ 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 $\psi_0(\ell) = \varphi_0(\ell) \cup (\emptyset, m_0, f_0)$ and $\psi_1(\ell) = \varphi_1(\ell) \cup (\emptyset, m_1, f_1)$ when an oracle is called on $m_\#$ under condition $f_\#$.

D. Invariant Synthesis

In order to find the initial pre-condition $\varphi_\#$ required to complete the whole proof by induction of Appendix E-A, we proceed iteratively, attempting to find an invariant $\varphi_\#$ as the result of a fixed-point computation. We start with the assertion $\varphi^0_\#$ stipulating that all logs are empty. At iteration i, we will have an assertion $\varphi^i_\#$, and we use our proof search procedure to complete induction-free derivations. If this succeeds, it yields several post-conditions $\psi_\#$, which can be regarded as a single post-condition by taking their union. We define $\varphi^{i+1}_\#$ 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 $\varphi^i_\#$ as post-conditions, can be obtained by weakening post-conditions to $\varphi^i_\#$. 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 [30]. 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. [31], [32].

Example 19. *In the proof of Example 17, starting with φ^0 which maps the two logs to $\emptyset$, the first derivation of our three inductive bi-deduction judgements yields, after some simplifications, $\varphi^1(\ell_{\mathsf{hash}}) = (\{t, i, j\}, \mathsf{n}\ (i, j), t = \mathsf{T}(i, j) < t_0)$ and $\varphi^1(\ell_{\mathsf{chal}}) = \emptyset$. 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 20. *We explain how the judgements obtained at the end of Example 17 can be derived. To derive the final judgement, recall that* $\mathit{output@t_0} = \mathsf{h}(\langle \mathsf{n}\ (i_0, j_0), \mathsf{att}(\mathit{frame@pred}\ t_0)\rangle, \mathsf{k}\ i_0)$. *Because* att *is adversarial and* $\mathit{frame@pred}\ t_0$ *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 $(\emptyset, \mathsf{n}, (i_0, j_0), \mathtt{T}_S, \top)$. To conclude, we have to call the hash oracle, which adds the constraint $(\emptyset, \mathsf{k}, i_0, \mathtt{T}^{\mathsf{glob}}_{\mathsf{G,key}})$, and requires that φ implies that the hashed message is not in $\ell_{\mathsf{hash}} \cup \ell_{\mathsf{chal}}$.*

Because the constraint systems $\mathcal{C}_i$ will eventually be combined with $\mathcal{C}'$ 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 $\mathcal{C}_2 = \emptyset$, by definition of the input function and because the frame at $\mathit{pred}\ t < t$ is available from w'. Similarly, we easily obtain the frame judgement, with $\mathcal{C}_1 = \emptyset$.

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:

$$(\mathcal{E}, t, i, j), \Theta\ \tilde{\wedge}\ [t = \mathsf{T}(i, j)]_{\mathrm{e}}, \mathcal{C}'_3, (\varphi, \varphi) \vdash w' \rhd \mathsf{h}(\langle \mathsf{n}\ (i, j), \mathit{input@t}\rangle, \mathsf{k}\ i)$$

Clearly, the hashed message can be bi-deduced with $(\{i, j\}, \mathsf{n}, (i, j), \mathtt{T}_S, \top) \in \mathcal{C}'_3$. To compute the hash per se, there are two cases:

- *Either $i_0 \neq i$, in which case the adversary can sample the key $\mathsf{k}\ i$ and compute the hash itself. For that case, we add $(\{i\}, \mathsf{k}\ i, \mathtt{T}_S, i_0 \neq i)$ 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 $(, \mathsf{n}, (i_0, j), \mathtt{T}_S, \top)$, 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 $\mathcal{C}_3 = \{(\{i, j\}, \mathsf{n}, (i, j), \top), (\emptyset, \mathsf{k}, i_0, \top)\}$.