Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2006-047 June 20, 2006

Using Task-Structured Probabilistic I/O Automata to Analyze an Oblivious Transfer Protocol

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

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Using Task-Structured Probabilistic I/O Automata to Analyze an Oblivious Transfer Protocol

Ran Canetti
MIT and IBM TJ Watson Research Center
canetti@theory.csail.mit.edu, canetti@watson.ibm.com

Ling Cheung
The University of Nijmegen
lcheung@cs.kun.nl

Dilsun Kaynar
MIT
dilsun@theory.csail.mit.edu

Moses Liskov
The College of William and Mary
mliskov@cs.wm.edu

Nancy Lynch
MIT
lynch@theory.csail.mit.edu

Olivier Pereira
Université catholique de Louvain
olivier.pereira@uclouvain.be

Roberto Segala
The University of Verona
roberto.segala@univr.it

June 19, 2006

Abstract

The Probabilistic I/O Automata framework of Lynch, Segala and Vaandrager provides tools for precisely specifying protocols and reasoning about their correctness using multiple levels of abstraction, based on implementation relationships between these levels. We enhance this framework to allow analyzing protocols that use cryptographic primitives. This requires resolving and reconciling issues such as nondeterministic behavior and scheduling, randomness, resource-bounded computation, and computational hardness assumptions. The enhanced framework allows for more rigorous and systematic analysis of cryptographic protocols. To demonstrate the use of this framework, we present an example analysis that we have done for an Oblivious Transfer protocol.

Contents

List of Figures

1 Introduction

The task of modeling and analysis of cryptographic protocols is typically complex, involving many subtleties and details, even when the analyzed protocols are simple. This causes security analysis of cryptographic protocols to be susceptible to errors and omissions. Our goal is to present a method for analyzing cryptographic protocols rigorously and systematically, while taking into account computational issues regarding cryptographic primitives.

Our approach is based on an extension of the Probabilistic I/O Automata (PIOA) framework developed in the concurrency semantics research community [Seg95, LSV03]. We represent protocols as probabilistic I/O automata, which are essentially interacting state machines, and use the associated modeling and proof techniques that have proved to be useful in the analysis of distributed algorithms that use randomization [Seg97, PSL00].

Briefly, a PIOA is a kind of abstract automaton. It includes *states*, *start states*, and *actions*. Each action has an associated set of *transitions*, which go from states to probability distributions on states. PIOAs are capable of expressing both *probabilistic choices* and *nondeterministic choices*. PIOAs that model individual components of a system may be composed to yield a PIOA model for the entire system. Many interesting properties of systems described using PIOAs can be expressed as *invariant assertions*, that is, properties of the system state that are true in all reachable states. Such properties are proved by induction on the length of an execution. The PIOA framework also supports the description of systems at multiple levels of abstraction. It includes notions of *implementation*, which assert that a "low-level" system is indistinguishable from another, "higher-level" system, from the point of view of some common "environment". The framework also includes various kinds of *simulation relations*, which provide sufficient conditions for proving implementation relationships between systems.

We think that the support for a combination of nondeterministic and probabilistic choices is a significant feature of the PIOA framework that allows generality and simplicity in modeling. We prefer not to be forced to specify results of choices that are not needed for achieving specific correctness or security guarantees: unnecessary specification not only restricts generality, but also introduces "clutter" into models that can obscure the reasons a system works correctly. This work demonstrates how nondeterministic and probabilistic choices can be reconciled in a cryptographic setting that provides security only against resource-bounded adversaries.

We formulate the security of a protocol based on the notion of "realizing an ideal functionality" within the universally composable (UC) security framework [Can01]. In the UC framework, the functionality that is to be achieved by a protocol is described as an ideal process, a kind of trusted party that computes the correct result from given inputs. A protocol is defined to be secure if for any adversary $\mathcal{A}$ that interacts with the protocol, there exists a "simulator" $\mathcal{S}$ that interacts with the ideal process such that no external environment can distinguish whether it is interacting with the protocol and $\mathcal{A}$ or, with the ideal process and $\mathcal{S}$. In the PIOA framework we formalize this notions as follows. We represent both the protocol and its specification (which is the composition of the ideal process and the simulator) as PIOAs, and we say that the protocol satisfies both its correctness and security requirements if it implements its specification. (The existential quantifier over the simulator in the UC definition is captured via appropriate nondeterministic choices available to the simulator component.)

As an essential part of the analysis, we model computationally-bounded adversaries, and restrict attention to such adversaries. In the same vein, a protocol is considered secure if the success probability of an attack on the protocol is negligible. Furthermore, as typical in cryptographic protocols, the security claims are conditional: they guarantee security only under computational hardness assumptions. Hence, to represent the computationally-bounded components we need special kinds of PIOAs, namely, PIOAs whose executions are polynomial-time-bounded. Moreover, we have to reconcile issues regarding nondeterminism and scheduling, time-bounded computation, and computational hardness assumptions. We list below the major extensions we made to the existing PIOA theory to be able to use the traditional reasoning techniques of the PIOA framework in the analysis of cryptographic protocols and to be able express computational assumptions.

(1) Resolution of nondeterminism: In the original PIOA framework [Seg95, LSV03] the next transition is chosen as a function of the entire past execution. This gives schedulers too much power in the sense that a scheduler can make choices based on supposedly secret information within non-adversarial components. This would provide a way of "leaking secrets" to the adversarial components. We extended the PIOA framework with a new notion of *tasks* (an equivalence relation on actions) and we describe the scheduler as simply an arbitrary sequence of tasks. For example, a scheduler can specify that the next task in a security protocol is to "send the round 1 message". Then the protocol participant that is responsible for sending the round 1 message can determine from its state exactly what round 1 message it should send, or if it should send no round 1 message at all. This is ensured by means of certain restrictions imposed on task-PIOAs (see Section 3.1).

(2) Simulation relations: We defined a new kind of *simulation relation*, which allows one to establish a correspondence between probability distributions of executions at two levels of abstraction, and which allows splitting of distributions in order to show that individual steps preserve the correspondence.

(3) Time-bounded PIOAs: We developed a new theory for *time-bounded PIOAs*, which impose time bounds on the individual steps of the PIOAs; and a new *approximate, time-bounded, implementation relationship* between time-bounded PIOAs that uses time-bounded task schedulers. This notion of implementation is sufficient to capture the typical relationships between cryptographic primitives and the abstractions they are supposed to implement.

Example. We exemplify our approach by analyzing an Oblivious Transfer (OT) protocol. In particular, we analyze the classic protocol of [EGL85, GMW87], which uses trap-door permutations (and hard-core predicates for them) as the underlying cryptographic primitive. The protocol is designed to achieve the oblivious transfer functionality, that is, a transmitter T sends two bits (x_0, x_1) to a receiver R who decides to receive only one of them, while preventing T from knowing which one was delivered. We focus on realizing OT in the presence of a passive adversary where even the corrupted parties follow the protocol instructions. Furthermore, we assume non-adaptive corruptions, where the set of corrupted parties is fixed before the protocol execution starts.

Even though the analyzed protocol and functionality are relatively simple, our exercise is interesting. OT is a powerful primitive that has been shown to be complete for multi-party secure protocols, in the sense that one can construct protocols for securely realizing any functionality using OT as the only cryptographic primitive. It is also interesting because it imposes secrecy requirements when either party is corrupted, in addition to correctness requirements. The protocol uses cryptographic primitives and computational hardness assumptions in an essential way and the analysis presents one with issues that need to be resolved to be able to establish general modeling idioms and verification methods that can be used in cryptographic analysis of *any* cryptographic protocol.

At a very high-level the analysis proceeds as follows. We define two PIOAs that represent the "real system", which captures the protocol execution, and the "ideal system", which captures the ideal specification for OT. We show that the real system implements the ideal system with respect to the notion of approximate, time-bounded, implementation. The complete proof would consist of four cases, depending on the set of parties that are corrupted. When only the transmitter T is corrupted, and when both parties are corrupted, it is possible to show that the real system implements the ideal system unconditionally. However, when neither party is corrupted or when the receiver is corrupted, implementation can be shown only in a "computational sense". We present here only the case where the receiver R is corrupted, since we believe this is the most interesting case. The proof involves relatively more subtle interactions of the automaton representing the functionality and the simulator; the simulator needs to receive the selected bit from the functionality to be able to output the correct value.

Following the usual proof methods for distributed algorithms, we decompose our proofs into several stages, with general transitivity results used to combine the results of the stages. Specifically, we use a hierarchy of four automata, where the protocol automaton, which uses hard-core predicates lies at the

lowest level and the specification, which uses their random counterparts lies at the highest level. We use two intermediate-level automata, which in two stages replace all the uses of hard-core predicates by random bits. A feature of our proofs is that complicated reasoning about particular cryptographic primitives—in this case, a hard-core predicate—is isolated to a single stage of the proof, the proof that shows the implementation relationship between the two intermediate-level automata. This is the only stage of the proof that uses our new approximate implementation relationship. For other stages we use the perfect implementation relationship.

Preliminary version. In our initial versions of the Oblivious Transfer proof, described in Technical Reports [CCK+05], the PIOA model used in the analysis has more restrictions than what we report here—in fact, enough to rule out branching behavior of the sort that is needed for describing adaptive adversarial schedulers. This restriction was undesirable. Therefore, we revisited some of the definitions in that work to make our model more general. In [CCK+05] we present proofs for four cases, one for each possible value for the set of corrupted parties. We have redone the proof for the case where only the receiver is corrupted assuming the new generalized definitions. We think that redoing the remaining three cases will be relatively straightforward.

Related work. Several papers have recently proposed the direct mechanization and formalization of concrete cryptographic analysis of protocols, in a number of different contexts. Examples include representing analysis as a sequence of games [Sho04, BR04], as well as methods for further formalizing that process [Bla05, Hal05], or automated proof checking techniques [BCT04, Tar05]. Our work differs from those in two main respects. First, those papers do not address ideal-process-based notion of security, namely they do not address asserting that a protocol realizes a specification process in a standard cryptographic sense, and hence do not provide any secure composability guarantees. In contrast, our analytical framework provides strong composability guarantees in a natural way. Furthermore, our analysis enjoys the extra rigor and detail that underly the PIOA framework.

There are other formal frameworks for analyzing cryptographic protocols that have similar motivations to ours such as those of Backes, Pfitzmann, and Waidner [PW00, BPW04b] (based on a variant of probabilistic I/O automata), Lincoln et al. [LMMS98, RMST04] (based on process-algebra), and Canetti [Can01] (based on interactive Turing machines). These frameworks use different mathematical objects and techniques for modeling and analyzing cyrptographic protocols, each of which offers particular strengths. A key goal of our work has been to support simplicity and generality in modeling; we allow the use of nondeterminism in modeling all kinds of choices that are inconsequential in achieving correctness or security guarantees, not only in modeling uncertainties in the external environment. We also provide a simple new scheduling mechanism for resolving nondeterminism, based on the notion of tasks (equivalence relations on actions). We remark that in proving security properties, we do not need to resolve nondeterminism beforehand to show that a protocol implements a specification. Our implementation definitions quantify over all possible task schedules. A more detailed comparison between task-PIAOs and the asynchronous reactive systems of Backes et al. is available in Appendix A. A similar comparison with the probabilistic polynomial-time process calculus of Lincoln et al. should be available shortly.

Roadmap. In Section 2 we give the preliminary mathematical definitions that are used in subsequent sections and define PIOAs, which are basic elements of the formal model presented in this paper. In Section 3 we introduce our basic theory of task-PIOAs, including the definitions of implementation and simulation relation. In Section 4 we define time-bounded task-PIOAs and extend our previous results about task-PIOAs to time-bounded task-PIOAs. We also define polynomial-time-bounded task-PIOA families. In Section 6, we show how a classical computational definition, namely the definition of hard-core predicates for trap-door permutations, can be expressed in terms of polynomial-time-bounded task-PIOA families, and prove the equivalence of the two definitions.

We start presenting our example analysis in Section 7, which describes how we model the ideal system. Then, in Section 5, we describe random source automata, which we will use in our task-PIOA

model of the protocol (real system) presented in Section 8. In Section 9, we explain the main security theorems we prove, these proofs being detailed in Section 10.

2 Mathematical Foundations

2.1 Sets, functions etc.

We write $\mathsf{R}^{\geq 0}$ and R^{+} for the sets of nonnegative real numbers and positive real numbers, respectively.

If X is any set, then we denote the set of finite sequences and infinite sequences of elements from X by X^* and X^ω, respectively. If ρ is a sequence then we use $|\rho|$ to denote the length of ρ. We use λ to denote the empty sequence (over any set).

If R is an equivalence relation over a set X, then we write $x \equiv_R x'$ provided that x and x' are in the same equivalence class. We sometimes write $S \in R$ if S is an equivalence class of R.

2.1.1 Probability measures

We present the basic definitions that we need for probability measures. We also define three operations involving probability measures: flattening, lifting, and expansion. We use these in defining a new kind of simulation relation for task-PIOAs, in Section 3.8. All of these have been defined elsewhere, for example, [LSV03, JL91].

Basic definitions: A σ-field over a set X is a set $\mathcal{F} \subseteq 2^X$ that contains the empty set and is closed under complement and countable union. A pair $(X, \mathcal{F})$ where $\mathcal{F}$ is a σ-field over X, is called a *measurable space*. A measure on a measurable space $(X, \mathcal{F})$ is a function $\mu : \mathcal{F} \to [0, \infty]$ that is countably additive: for each countable family $\{X_i\}_i$ of pairwise disjoint elements of $\mathcal{F}$, $\mu(\cup_i X_i) = \sum_i \mu(X_i)$. A *probability measure* on $(X, \mathcal{F})$ is a measure on $(X, \mathcal{F})$ such that $\mu(X) = 1$. A *sub-probability measure* on $(X, \mathcal{F})$ is a measure on $(X, \mathcal{F})$ such that $\mu(X) \leq 1$.

A *discrete probability measure* on a set X is a probability measure μ on $(X, 2^X)$, such that, for each $C \subseteq X$, $\mu(C) = \sum_{c \in C} \mu(\{c\})$. A *discrete sub-probability measure* on a set X, is a sub-probability measure μ on $(X, 2^X)$, such that for each $C \subseteq X$, $\mu(C) = \sum_{c \in C} \mu(\{c\})$. We define $Disc(X)$ and $SubDisc(X)$ to be, respectively, the set of discrete probability measures and discrete sub-probability measures on X. In the sequel, we often omit the set notation when we denote the measure of a singleton set.

A *support* of a probability measure μ is a measurable set C such that $\mu(C) = 1$. If μ is a discrete probability measure, then we denote by $supp(\mu)$ the set of elements that have non-zero measure; $supp(\mu)$ is a support of μ. We let $\delta(x)$ denote the *Dirac measure* for x, the discrete probability measure that assigns probability 1 to $\{x\}$.

If $\{\rho_i\}_{i \in I}$ be a countable family of measures on $(X, \mathcal{F}_X)$, and $\{p_i\}_{i \in I}$ is a family of non-negative values, then the expression $\sum_{i \in I} p_i \rho_i$ denotes a measure ρ on $(X, \mathcal{F}_X)$ such that, for each $C \in \mathcal{F}_X$, $\rho(C) = \sum_{i \in I} p_i \rho_i(C)$.

Given two discrete measures μ_1, μ_2 on $(X, 2^X)$ and $(Y, 2^Y)$, respectively, we denote by $\mu_1 \times \mu_2$ the *product measure*, that is, the measure on $(X \times Y, 2^{X \times Y})$ such that $\mu_1 \times \mu_2(x, y) = \mu_1(x) \times \mu_2(y)$ for each $x \in X$, $y \in Y$.

A function $f : X \to Y$ is said to be measurable from $(X, \mathcal{F}_X) \to (Y, \mathcal{F}_Y)$ if the inverse image of each element of $\mathcal{F}_Y$ is an element of $\mathcal{F}_X$, that is, for each $C \in \mathcal{F}_Y$, $f^{-1}(C) \in \mathcal{F}_X$. In such a case, given a measure μ on $(X, \mathcal{F}_X)$, the function $f(\mu)$ defined on $\mathcal{F}_Y$ by $f(\mu)(C) = \mu(f^{-1}(C))$ for each $C \in \mathcal{Y}$ is a measure on $(Y, \mathcal{F}_Y)$ and is called the *image measure* of μ under f.

y more.

Flattening: The first operation, which we call "flattening", takes a discrete probability measure over probability measures and "flattens" it into a single probability measure.

Let η be a discrete probability measure on $Disc(X)$. Then the flattening of η, denoted by $flatten(\eta)$, is the discrete probability measure on X defined by $flatten(\eta) = \sum_{\mu \in Disc(X)} \eta(\mu)\mu$.

Lemma 2.1 *Let η be a discrete probability measure on $Disc(X)$ and let f be a function from X to Y. Then $f(flatten(\eta)) = flatten(f(\eta))$.*

Proof. By the definition of flattening, $f(flatten(\eta)) = f(\sum_{\mu \in Disc(X)} \eta(\mu)\mu)$. By distributing f, we obtain that this is equal to $\sum_{\mu \in Disc(X)} \eta(\mu) f(\mu)$. By rearranging terms in this last expression, we obtain that $f(flatten(\eta)) = \sum_{\sigma \in Disc(Y)} \sum_{\mu \in f^{-1}(\sigma)} \eta(\mu)\sigma$. Now, $\sum_{\mu \in f^{-1}(\sigma)} \eta(\mu) = f(\eta)(\sigma)$, which implies that $f(flatten(\eta)) = \sum_{\sigma \in Disc(Y)} f(\eta)(\sigma)\sigma$. But the right-hand expression is the definition of $flatten(f(\eta))$, as needed. □

Lemma 2.2 *Let $\{\eta_i\}_{i \in I}$ be a countable family of measures on $Disc(X)$, and let $\{p_i\}_{i \in I}$ be a family of probabilities such that $\sum_{i \in I} p_i = 1$. Then $flatten(\sum_{i \in I} p_i \eta_i) = \sum_{i \in I} p_i flatten(\eta_i)$.*

Lifting: The second operation, which we call "lifting", takes a relation between two domains X and Y and "lifts" it to a relation between discrete measures over X and Y. We allow the correspondence to be rather general: we express it in terms of the existence of a weighting function on elements of $X \times Y$ that can be used to relate the two measures.

Let R be a relation from X to Y. The *lifting* of R, denoted by $\mathcal{L}(R)$, is a relation from $Disc(X)$ to $Disc(Y)$ such that $\mu_1 \; \mathcal{L}(R) \; \mu_2$ iff there exists a function $w : X \times Y \to \mathbf{R}^{\geq 0}$, called a *weighting function*, such that

1. for each $x \in X$ and $y \in Y$, $w(x, y) > 0$ implies $x \; R \; y$,
2. for each $x \in X$, $\sum_y w(x, y) = \mu_1(x)$, and
3. for each $y \in Y$, $\sum_x w(x, y) = \mu_2(y)$.

Expansion: Finally, we have the third operation, the "expansion" operation, which is the one we use directly in our new definition of simulation relations. The expansion of a relation R relates a measure on X to a measure on Y provided that the two measures can be "expanded" into corresponding measures on measures. Here, the correspondence between the two measures on measures is rather general, in fact, we express it in terms of the lifting operation.

Let R be a relation from $Disc(X)$ to $Disc(Y)$. The *expansion* of R, denoted by $\mathcal{E}(R)$, is the relation from $Disc(X)$ to $Disc(Y)$ such that $\mu_1 \; \mathcal{E}(R) \; \mu_2$ iff there exist two discrete measures η_1 and η_2 on $Disc(X)$ and $Disc(Y)$, respectively, such that

1. $\mu_1 = flatten(\eta_1)$,
2. $\mu_2 = flatten(\eta_2)$, and
3. $\eta_1 \; \mathcal{L}(R) \; \eta_2$.

The following lemma provides an equivalent characterization of the expansion relation:

Lemma 2.3 *Let R be a relation on $Disc(X) \times Disc(Y)$. Then $\mu_1 \; \mathcal{E}(R) \; \mu_2$ iff there exists a countable index set I, a discrete probability measure p on I, and two collections of probability measures $\{\mu_{1,i}\}_I$,$\{\mu_{2,i}\}_I$ such that*

1. $\mu_1 = \sum_{i \in I} p(i)\mu_{1,i}$,
2. $\mu_2 = \sum_{i \in I} p(i)\mu_{2,i}$, *and*
3. *for each $i \in I$, $\mu_{1,i} \; R \; \mu_{2,i}$.*

Proof. Let $\mu_1\ \mathcal{E}(R)\ \mu_2$, and let η_1, η_2 and w be the measures and weighting functions used in the definition of $\mathcal{E}(R)$. Let $\{(\mu_{1,i}, \mu_{2,i})\}_{i\in I}$ be an enumeration of the pairs for which $w(\mu_{1,i}, \mu_{2,i}) > 0$, and let $p(i)$ be $w(\mu_{1,i}, \mu_{2,i})$. Then p, $\{(\mu_{1,i})\}_{i\in I}$, and $\{(\mu_{2,i})\}_{i\in I}$ satisfy Items 1, 2, and 3.

Conversely, given p, $\{(\mu_{1,i})\}_{i\in I}$, and $\{(\mu_{2,i})\}_{i\in I}$, define $\eta_1(\mu)$ to be $\sum_{i|\mu=\mu_{1,i}} p(i)$, $\eta_2(\mu)$ to be $\sum_{i|\mu=\mu_{2,i}} p(i)$, and define $w(\mu_1', \mu_2')$ to be $\sum_{i|\mu_1'=\mu_{1,i},\mu_2'=\mu_{2,i}} p(i)$. Then, η_1, η_2 and w satisfy the properties required in the definition of $\mathcal{E}(R)$. □

The next, rather technical lemma gives us a sufficient condition for showing that a pair of functions, f and g, transforms $\mathcal{E}(R)$-related probability measures μ_1 and μ_2 to other $\mathcal{E}(R)$-related probability measures. The required condition is that f and g convert each pair ρ_1, ρ_2 of R-related probability measures witnessing that $\mu_1\ \mathcal{E}(R)\ \mu_2$ to $\mathcal{E}(R)$-related probability measures. We will use this lemma in the soundness proof for our new kind of simulation relation, in Lemma 3.28; there, the two functions f and g apply corresponding sequences of tasks to corresponding measures on states.

Lemma 2.4 *Let R be a relation from $Disc(X)$ to $Disc(Y)$, and let f, g be two endo-functions on $Disc(X)$ and $Disc(Y)$, respectively, that distribute over convex combinations of measures, that is, for each countable family $\{\rho_i\}_i$ of discrete measures on X and each countable family of probabilities $\{p_i\}_i$ such that $\sum_i p_i = 1$, $f(\sum_i p_i\rho_i) = \sum_i p_i f(\rho_i)$, and similarly, for each countable family $\{\rho_i\}_i$ of discrete measures on Y and each countable family of probabilities $\{p_i\}_i$ such that $\sum_i p_i = 1$, $g(\sum_i p_i\rho_i) = \sum_i p_i g(\rho_i)$. Let μ_1 and μ_2 be two measures on X and Y respectively, such that $\mu_1\ \mathcal{E}(R)\ \mu_2$, and let η_1, η_2, and w be a pair of measures and a weighting function witnessing that $\mu_1\ \mathcal{E}(R)\ \mu_2$. Suppose further that, for any two distributions $\rho_1 \in supp(\eta_1)$ and $\rho_2 \in supp(\eta_2)$ such that $w(\rho_1, \rho_2) > 0$, $f(\rho_1)\ \mathcal{E}(R)\ g(\rho_2)$.*
Then $f(\mu_1)\ \mathcal{E}(R)\ g(\mu_2)$.

Proof. For each $\rho_1 \in supp(\eta_1)$ and $\rho_2 \in supp(\eta_2)$ such that $w(\rho_1, \rho_2) > 0$, let $(\eta_1)_{\rho_1,\rho_2}$, $(\eta_2)_{\rho_1,\rho_2}$, and $w_{\rho_1\rho_2}$ be a pair of measures and a weighting function that prove that $f(\rho_1)\ \mathcal{E}(R)\ g(\rho_2)$. We know that these are well-defined since, by assumption, $f(\rho_1)\ \mathcal{E}(R)\ g(\rho_2)$ whenever $w(\rho_1, \rho_2) > 0$. Let W denote the set of pairs (ρ_1, ρ_2) such that $w(\rho_1, \rho_2) > 0$.

Let $\eta_1' = \sum_{(\rho_1,\rho_2)\in W} w(\rho_1, \rho_2)(\eta_1)_{\rho_1,\rho_2}$ and let $\eta_2' = \sum_{(\rho_1,\rho_2)\in W} w(\rho_1, \rho_2)(\eta_2)_{\rho_1,\rho_2}$. Let $w' = \sum_{(\rho_1,\rho_2)\in W} w(\rho_1, \rho_2)w_{\rho_1,\rho_2}$.

We show that η_1', η_2', and w' prove that $f(\mu_1)\ \mathcal{E}(R)\ g(\mu_2)$.

1. $f(\mu_1) = flatten(\eta_1')$.

   By definition of η_1', $flatten(\eta_1') = flatten(\sum_{(\rho_1,\rho_2)\in W} w(\rho_1, \rho_2)(\eta_1)_{\rho_1,\rho_2})$. By Lemma 2.2, this is in turn equal to $\sum_{(\rho_1,\rho_2)\in W} w(\rho_1, \rho_2) flatten((\eta_1)_{(\rho_1,\rho_2)})$. By definition of $(\eta_1)_{(\rho_1,\rho_2)}$, we know that $flatten((\eta_1)_{(\rho_1,\rho_2)}) = f(\rho_1)$, so we obtain that $flatten(\eta_1') = \sum_{(\rho_1,\rho_2)\in W} w(\rho_1, \rho_2) f(\rho_1)$.

   We claim that the right side is equal to $f(\mu_1)$: Since $\mu_1 = flatten(\eta_1)$, by the definition of flattening, $\mu_1 = \sum_{\rho_1\in Disc(X)} \eta_1(\rho_1)\rho_1$. Then, by distributivity of f, $f(\mu_1) = \sum_{\rho_1\in Disc(X)} \eta_1(\rho_1) f(\rho_1)$. By definition of lifting, $\eta_1(\rho_1) = \sum_{\rho_2\in Disc(Y)} w(\rho_1, \rho_2)$.
   Therefore, $f(\mu_1) = \sum_{\rho_1\in Disc(X)} \sum_{\rho_2\in Disc(Y)} w(\rho_1, \rho_2) f(\rho_1)$, and this last expression is equal to $\sum_{(\rho_1,\rho_2)\in W} w(\rho_1, \rho_2) f(\rho_1)$, as needed.

2. $g(\mu_2) = flatten(\eta_2')$.

   Analogous to the previous case.

3. $\eta_1'\ \mathcal{L}(R)\ \eta_2'$ using w' as a weighting function.

   We verify that w' satisfies the three conditions in the definition of a weighting function:

   (a) Let ρ_1', ρ_2' be such that $w'(\rho_1', \rho_2') > 0$. Then, by definition of w', there exists at least one pair $(\rho_1, \rho_2) \in R$ such that $w_{\rho_1,\rho_2}(\rho_1', \rho_2') > 0$. Since w_{ρ_1,ρ_2} is a weighting function, $\rho_1'\ R\ \rho_2'$ as needed.

(b) By definition of w', $\sum_{\rho_2' \in Disc(Y)} w'(\rho_1', \rho_2') = \sum_{\rho_2' \in Disc(Y)} \sum_{(\rho_1,\rho_2)} w(\rho_1,\rho_2) w_{\rho_1,\rho_2}(\rho_1', \rho_2')$. By rearranging sums and using the fact that w_{ρ_1,ρ_2} is a weighting function, we obtain that $\sum_{\rho_2' \in Disc(Y)} w'(\rho_1', \rho_2') = \sum_{(\rho_1,\rho_2)} w(\rho_1,\rho_2)(\eta_1)_{\rho_1,\rho_2}(\rho_1')$. (Specifically, this uses the fact that $\sum_{\rho_2' \in Disc(Y)} w_{\rho_1,\rho_2}(\rho_1', \rho_2') = (\eta_1)_{\rho_1,\rho_2}(\rho_1')$.) This suffices since the right-hand side is the definition of $\eta_1'(\rho_1')$.

(c) Symmetric to the previous case.

□

2.2 Probabilistic I/O Automata

The definition of a PIOA is standard. A PIOA has states, a unique start state, and a set of actions, partitioned into input, output, and internal actions. It also has a set of "transitions", which are triples consisting of a state, an action, and a discrete distribution on next states. Note that a PIOA may exhibit both nondeterministic and probabilistic choices. Nondeterminism appears in the choice of the next transition to perform. Probabilistic choice occurs only in the choice of the next state, when a particular transition is performed.

Definition 2.5 *A* probabilistic I/O automaton (PIOA) *is a tuple* $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$*, where*

- Q *is a countable set of* states*,*
- $\bar{q} \in Q$ *is a* start state*,*
- I *is a countable set of* input actions*,*
- O *is a countable set of* output actions*,*
- H *is a countable set of* internal (hidden) actions*, and*
- $D \subseteq (Q \times (I \cup O \cup H) \times Disc(Q))$ *is a* transition relation.

We write A *for* $I \cup O \cup H$ *and refer to* A *as the* actions *of* $\mathcal{P}$*. We write* E *for* $I \cup O$ *and we refer to* E *as the* external actions *of* $\mathcal{P}$*. We assume that PIOA* $\mathcal{P}$ *satisfies the following conditions.*

1. I, O *and* H *are disjoint sets.*
2. ***Input enabling:*** *For every state* $q \in Q$ *and every action* $a \in I$*,* D *contains some triple of the form* (q, a, μ)*.*
3. ***Next-transition determinism:*** *For every state* q *and action* a*, there is at most one transition of the form* (q, a, μ)*. We write* $tr_{q,a}$ *to denote this transition, and* $\mu_{q,a}$ *to denote the target measure of this transition, if the transition exists. (Otherwise, these notations are undefined.)*

We say that an action a *is* enabled in *a state* q *if* D *contains a transition* (q, a, μ) *for some* μ*.*

Note that the next-transition determinism and the countability of Q, I, O, and H are restrictions that are not present in earlier definitions of probabilistic automata [LSV03]. We introduce these in the interests of simplicity. Input-enabling is standard.

We denote the elements of an automaton $\mathcal{P}$ by $Q_{\mathcal{P}}, \bar{q}_{\mathcal{P}}, I_{\mathcal{P}}, O_{\mathcal{P}}, H_{\mathcal{P}}, D_{\mathcal{P}}, A_{\mathcal{P}}$ and $E_{\mathcal{P}}$. Often we use the generic name $\mathcal{P}$ for a generic automaton; in this case we omit the subscripts, writing simply $Q, \bar{q}, I, O, H, D, A$ and E.

An *execution fragment* of a PIOA $\mathcal{P}$ is a finite or infinite sequence $\alpha = q_0\, a_1\, q_1\, a_2\, \ldots$ of alternating states and actions, starting with a state and, if the sequence is finite ending in a state, where for each (q_i, a_{i+1}, q_{i+1}) there exists a transition $(q_i, a_{i+1}, \mu) \in D$ with $q_{i+1} \in supp(\mu)$. If α is a finite sequence, then the last state of α is denoted by $lstate(\alpha)$. If α is an execution fragment of $\mathcal{P}$ and a is an action of $\mathcal{P}$ that is enabled in $lstate(\alpha)$, then we write $tr_{\alpha,a}$ as an abbreviation for $tr_{lstate(\alpha),a}$.

An *execution* of $\mathcal{P}$ is an execution fragment whose first state is the start state $\bar{q}$. We let $frags(\mathcal{P})$ and $frags^*(\mathcal{P})$ denote, respectively, the set of all execution fragments and the set of finite execution fragments of $\mathcal{P}$. Similarly, we let $execs(\mathcal{P})$ and $execs^*(\mathcal{P})$ denote, respectively, the set of all executions and the set of finite executions of $\mathcal{P}$.

The *trace* of an execution fragment α of an automaton $\mathcal{P}$, written $trace(\alpha)$, is the sequence obtained by restricting α to the set of external actions of $\mathcal{P}$. We say that β is a *trace* of automaton $\mathcal{P}$ if there is an execution α of $\mathcal{P}$ with $trace(\alpha) = \beta$.

2.2.1 σ-fields of execution fragments and traces

In order to talk about probabilities for executions and traces of a PIOA, we need appropriate σ-fields. We define a σ-field over the set of execution fragments of a PIOA $\mathcal{P}$:

Definition 2.6 *The* cone *of a finite execution fragment α, denoted by C_α, is the set $\{\alpha' \in frags(\mathcal{P}) \mid \alpha \leq \alpha'\}$. Then $\mathcal{F}_\mathcal{P}$ is the σ-field generated by the set of cones of finite execution fragments of $\mathcal{P}$.*

Observe that, since Q, I, O, and H are countable, the set of finite execution fragments of $\mathcal{P}$ is countable, and hence the set of cones of finite execution fragments of $\mathcal{P}$ is countable. Therefore, any union of cones is measurable. Observe also that, for each finite execution fragment α, the set $\{\alpha\}$ is measurable since it can be expressed as the intersection of C_α with the complement of $\cup_{\alpha':\alpha<\alpha'} C_{\alpha'}$. Thus, any set of finite execution fragments is measurable, or, in other words, the discrete σ-field of finite executions is included in $\mathcal{F}_\mathcal{P}$.

We often refer to a probability measure on the σ-field $\mathcal{F}_\mathcal{P}$ generated by cones of execution fragments of a PIOA $\mathcal{P}$ as simply a *probability measure on execution fragments of $\mathcal{P}$*.

In many places in this paper, we will want to talk about probability measures on finite execution fragments, rather than arbitrary execution fragments. Thus, we define:

Definition 2.7 *If ϵ is a probability measure on execution fragments of $\mathcal{P}$, then we say that ϵ is* finite *if the set of finite execution fragments is a support for ϵ.*

Since any set of finite execution fragments is measurable, any finite probability measure on execution fragments of $\mathcal{P}$ can also be viewed as a discrete probability measure on the set of finite execution fragments. Formally, given any finite probability measure ϵ on execution fragments of $\mathcal{P}$, we may define a discrete probability measure $finite(\epsilon)$ on the set of finite execution fragments of $\mathcal{P}$ by simply defining $finite(\epsilon)(\alpha) = \epsilon(\alpha)$ for every finite execution fragment α of $\mathcal{P}$. The difference between $finite(\epsilon)$ and ϵ is simply that the domain of ϵ is the set of all execution fragments of $\mathcal{P}$, whereas the domain of $finite(\epsilon)$ is the set of all *finite* executions of $\mathcal{P}$. Henceforth, we will ignore the distinction between $finite(\epsilon)$ and ϵ.

Definition 2.8 *Let ϵ and ϵ' be probability measures on execution fragments of PIOA $\mathcal{P}$. Then we say that ϵ is a* prefix *of ϵ', denoted by $\epsilon \leq \epsilon'$, if, for each finite execution fragment α of $\mathcal{P}$, $\epsilon(C_\alpha) \leq \epsilon'(C_\alpha)$.*

Definition 2.9 *A* chain *of probability measures on execution fragments of PIOA $\mathcal{P}$ is an infinite sequence, $\epsilon_1, \epsilon_2, \cdots$ of probability measures on execution fragments of $\mathcal{P}$ such that, for each $i \geq 0$, $\epsilon_i \leq \epsilon_{i+1}$. Given a chain $\epsilon_1, \epsilon_2, \ldots$ of probability measures on execution fragments of $\mathcal{P}$, we define a new function ϵ on the σ-field generated by cones of execution fragments of $\mathcal{P}$ as follows: For each finite execution fragment α,*

$$\epsilon(C_\alpha) = \lim_{i \to \infty} \epsilon_i(C_\alpha).$$

Standard measure theoretical arguments ensure that ϵ can be extended uniquely to a probability measure on the σ-field generated by the cones of finite execution fragments. Furthermore, for each $i \geq 0$, $\epsilon_i \leq \epsilon$. We call ϵ the limit *of the chain, and we denote it by $\lim_{i \to \infty} \epsilon_i$.*

If α is a finite execution fragment of a PIOA $\mathcal{P}$ and a is an action of $\mathcal{P}$, then $C_{\alpha a}$ denotes the set of execution fragments of $\mathcal{P}$ that start with αa.

The cone construction can also be used to define a σ-field of traces:

Definition 2.10 *The* cone *of a finite trace β, denoted by C_β, is the set $\{\beta' \in E^* \cup E^\omega \mid \beta \leq \beta'\}$, where $\leq$ denotes the prefix ordering on sequences. The σ-field of traces of $\mathcal{P}$ is simply the σ-field generated by the set of cones of finite traces of $\mathcal{P}$.*

Again, the set of cones is countable and the discrete σ-field on finite traces is included in the σ-field generated by cones of traces. We often refer to a probability measure on the σ-field generated by cones of traces of a PIOA $\mathcal{P}$ as simply a *probability measure on traces of $\mathcal{P}$*.

Definition 2.11 *If τ is a probability measure on traces of $\mathcal{P}$, then we say that τ is* finite *if the set of finite traces is a support for τ. Any finite probability measure on traces of $\mathcal{P}$ can also be viewed as a discrete measure on the set of finite traces.*

Definition 2.12 *Let τ and τ' be probability measures on traces of PIOA $\mathcal{P}$. Then we say that τ is a* prefix *of τ', denoted by $\tau \leq \tau'$, if, for each finite trace β of $\mathcal{P}$, $\tau(C_\beta) \leq \tau'(C_\beta)$.*

Definition 2.13 *A* chain *of probability measures on traces of PIOA $\mathcal{P}$ is an infinite sequence, $\tau_1, \tau_2, \cdots$ of probability measures on traces of $\mathcal{P}$ such that, for each $i \geq 0$, $\tau_i \leq \tau_{i+1}$. Given a chain $\tau_1, \tau_2, \ldots$ of probability measures on traces of $\mathcal{P}$, we define a new function τ on the σ-field generated by cones of traces of $\mathcal{P}$ as follows: For each finite trace β,*

$$\tau(C_\beta) = \lim_{i \to \infty} \tau_i(C_\beta).$$

Then τ can be extended uniquely to a probability measure on the σ-field of cones of finite traces. Furthermore, for each $i \geq 0$, $\tau_i \leq \tau$. We call τ the limit *of the chain, and we denote it by $\lim_{i \to \infty} \tau_i$.*

The *trace* function is a measurable function from the σ-field generated by cones of execution fragments of $\mathcal{P}$ to the σ-field generated by cones of traces of $\mathcal{P}$. If ϵ is a probability measure on execution fragments of $\mathcal{P}$ then we define the trace distribution of ϵ, $tdist(\epsilon)$, to be the image measure of ϵ under the function *trace*.

Lemma 2.14 *Let $\epsilon_1, \epsilon_2, \cdots$ be a chain of measures on execution fragments, and let ϵ be $\lim_{i \to \infty} \epsilon_i$. Then $\lim_{i \to \infty} tdist(\epsilon_i) = tdist(\epsilon)$.*

Proof. It suffices to show that, for any finite trace β, $\lim_{i \to \infty} tdist(\epsilon_i)(C_\beta) = tdist(\epsilon)(C_\beta)$. Fix a finite trace β.

Let Θ be the set of minimal execution fragments whose trace is in C_β. Then $trace^{-1}(C_\beta) = \cup_{\alpha \in \Theta} C_\alpha$, where all the cones are pairwise disjoint. Therefore, for $i \geq 0$, $tdist(\epsilon_i)(C_\beta) = \sum_{\alpha \in \Theta} \epsilon_i(C_\alpha)$, and $tdist(\epsilon)(C_\beta) = \sum_{\alpha \in \Theta} \epsilon(C_\alpha)$.

Since we have monote limits here (our limit are also supremums), limits commute with sums and our goal can be restated as showing: $\sum_{\alpha \in \Theta} \lim_{i \to \infty} \epsilon_i(C_\alpha) = \sum_{\alpha \in \Theta} \epsilon(C_\alpha)$. Since $\lim_{i \to \infty} \epsilon_i = \epsilon$, for each finite execution fragment α, $\lim_{i \to \infty} \epsilon_i(C_\alpha) = \epsilon(C_\alpha)$. Therefore, $\sum_{\alpha \in \Theta} \lim_{i \to \infty} \epsilon_i(C_\alpha) = \sum_{\alpha \in \Theta} \epsilon(C_\alpha)$, as needed. □

The *lstate* function is a measurable function from the discrete σ-field of finite execution fragments of $\mathcal{P}$ to the discrete σ-field of states of $\mathcal{P}$. If ϵ is a probability measure on execution fragments of $\mathcal{P}$, then we define the lstate distribution of ϵ, $lstate(\epsilon)$, to be the image measure of ϵ under the function *lstate*.

2.2.2 Probabilistic executions and trace distributions

Having established some groundwork in Section 2.2.1, we now define the specific probability measures on executions and traces that are generated by PIOAs. To define such probability measure, we must resolve the PIOA's nondeterminism. For this purpose, we define a "scheduler", which, after any finite execution fragment, selects the next transition:

Definition 2.15 *A* scheduler *for a PIOA $\mathcal{P}$ is a function $\sigma : frags^*(\mathcal{P}) \rightarrow SubDisc(D)$ such that $(q, a, \mu) \in supp(\sigma(\alpha))$ implies $q = lstate(\alpha)$.*

A scheduler σ describes what transitions to schedule after each finite execution fragment of $\mathcal{P}$. It associates sub-probability measures with finite execution fragments, which means that after a finite execution fragment α the probability $\sigma(\alpha)(D)$ may be strictly smaller than 1, or, in other words, that the scheduler σ terminates the computation after α with probability $1 - \sigma(\alpha)(D)$. As a notational convention we introduce a new symbol $\perp$ to denote termination, and we write $\sigma(\alpha)(\perp)$ to denote the probability $1 - \sigma(\alpha)(D)$ of terminating after α.

Definition 2.16 *A scheduler σ and a finite execution fragment α generate a measure $\epsilon_{\sigma,\alpha}$ on the σ-field generated by cones of execution fragments. The measure of a cone $C_{\alpha'}$ is defined recursively as follows:*

$$\epsilon_{\sigma,\alpha}(C_{\alpha'}) = \begin{cases} 0 & \text{if } \alpha' \not\leq \alpha \text{ and } \alpha \not\leq \alpha' \\ 1 & \text{if } \alpha' \leq \alpha \\ \epsilon_{\sigma,\alpha}(C_{\alpha''})\mu_{\sigma(\alpha'')}(a,q) & \text{if } \alpha' = \alpha''aq \text{ and } \alpha \leq \alpha'', \end{cases} \tag{1}$$

where $\mu_{\sigma(\alpha'')}(a, q)$ is the probability that $\sigma(\alpha'')$ gives a transition labeled by a and that the reached state is q. That is, $\mu_{\sigma(\alpha'')}(a, q) = \sigma(\alpha'')(tr_{\alpha'',a})\mu_{\alpha'',a}(q)$. Standard measure theoretical arguments ensure that $\epsilon_{\sigma,\alpha}$ is well-defined. We say that $\epsilon_{\sigma,\alpha}$ is generated *by σ and α. We call the state $fstate(\alpha)$ the* first state *of $\epsilon_{\sigma,\alpha}$ and denote it by $fstate(\epsilon_{\sigma,\alpha})$.*

If μ is a discrete probability measure over finite execution fragments, then we denote by $\epsilon_{\sigma,\mu}$ the measure $\sum_\alpha \mu(\alpha)\epsilon_{\sigma,\alpha}$ and we say that $\epsilon_{\sigma,\mu}$ is generated by *σ and μ. We call the measure $\epsilon_{\sigma,\mu}$ a* generalized probabilistic execution fragment *of $\mathcal{P}$.*

If $supp(\mu)$ contains only execution fragments consisting of a single state then we call $\epsilon_{\sigma,\mu}$ a probabilistic execution fragment *of $\mathcal{P}$. Finally, for the start state $\bar{q}$, we call $\epsilon_{\sigma,\bar{q}}$ a* probabilistic execution *of $\mathcal{P}$.*

The following lemmas give some simple equations expressing basic relationships involving the probabilities of various sets of execution fragments.

Lemma 2.17 *Let σ be a scheduler for PIOA $\mathcal{P}$, μ be a discrete probability measure on finite execution fragments of $\mathcal{P}$, and α be a finite execution fragment of $\mathcal{P}$. Then*

$$\epsilon_{\sigma,\mu}(C_\alpha) = \mu(C_\alpha) + \sum_{\alpha' < \alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha).$$

Proof. By definition of $\epsilon_{\sigma,\mu}$, $\epsilon_{\sigma,\mu}(C_\alpha) = \sum_{\alpha'} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha)$. Since, by definition, $\epsilon_{\sigma,\alpha'}(C_\alpha) = 1$ whenever $\alpha \leq \alpha'$, the equation above can be rewritten as $\epsilon_{\sigma,\mu}(C_\alpha) = \sum_{\alpha':\alpha\leq\alpha'} \mu(\alpha') + \sum_{\alpha'<\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha)$. Observe that $\sum_{\alpha':\alpha\leq\alpha'} \mu(\alpha') = \mu(C_\alpha)$. Thus, by substitution, we get the statement of the lemma. □

Lemma 2.18 *Let σ be a scheduler for PIOA $\mathcal{P}$, μ be a discrete probability measure on finite execution fragments of $\mathcal{P}$, and α be a finite execution fragment of $\mathcal{P}$. Then*

$$\epsilon_{\sigma,\mu}(C_\alpha) = \mu(C_\alpha - \{\alpha\}) + \sum_{\alpha' \leq \alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha).$$

Proof. Follows directly from Lemma 2.17 after observing that $\epsilon_{\sigma,\alpha}(C_\alpha) = 1$. □

Lemma 2.19 *Let σ be a scheduler for PIOA $\mathcal{P}$, and μ be a discrete measure on finite execution fragments of $\mathcal{P}$. Let $\alpha = \tilde{\alpha}aq$ be a finite execution fragment of $\mathcal{P}$. Then*

$$\epsilon_{\sigma,\mu}(C_\alpha) = \mu(C_\alpha) + (\epsilon_{\sigma,\mu}(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\,\sigma(\tilde{\alpha})(tr_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q).$$

Proof. By Lemma 2.17, by definition of $\epsilon_{\sigma,\alpha'}(C_\alpha)$, and by definition of $\mu_{\sigma(\tilde{\alpha})}(a,q)$, $\epsilon_{\sigma,\mu}(C_\alpha) = \mu(C_\alpha) + \sum_{\alpha'<\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_{\tilde{\alpha}})\sigma(\tilde{\alpha})(tr_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q)$. Observe that the factor $\sigma(\tilde{\alpha})(tr_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q)$ is a constant with respect to α', and thus can be moved out of the sum, so
$\epsilon_{\sigma,\mu}(C_\alpha) = \mu(C_\alpha) + (\sum_{\alpha'<\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_{\tilde{\alpha}}))(\sigma(\tilde{\alpha})(tr_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q))$. Since $\alpha' \leq \tilde{\alpha}$ if and only if $\alpha' < \alpha$, this yields $\epsilon_{\sigma,\mu}(C_\alpha) = \mu(C_\alpha) + (\sum_{\alpha'\leq\tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_{\tilde{\alpha}}))(\sigma(\tilde{\alpha})(tr_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q))$.

It suffices to show that $\sum_{\alpha'\leq\tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_{\tilde{\alpha}}) = \epsilon_{\sigma,\mu}(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$. But this follows immediately from Lemma 2.18 (with α instantiated as $\tilde{\alpha}$). □

Lemma 2.20 *Let σ be a scheduler for PIOA $\mathcal{P}$, μ be a discrete probability measure on finite execution fragments of $\mathcal{P}$, and α be a finite execution fragment of $\mathcal{P}$. Then*

$$\epsilon_{\sigma,\mu}(\alpha) = (\epsilon_{\sigma,\mu}(C_\alpha) - \mu(C_\alpha - \{\alpha\}))(\sigma(\alpha)(\bot)).$$

Proof. By definition of $\epsilon_{\sigma,\mu}$, $\epsilon_{\sigma,\mu}(\alpha) = \sum_{\alpha'} \mu(\alpha')\epsilon_{\sigma,\alpha'}(\alpha)$. The sum can be restricted to $\alpha' \leq \alpha$ since for all other α', $\epsilon_{\sigma,\alpha'}(\alpha) = 0$. Then, since for each $\alpha' \leq \alpha$, $\epsilon_{\sigma,\alpha'}(\alpha) = \epsilon_{\sigma,\alpha'}(C_\alpha)\sigma(\alpha)(\bot)$, we derive $\epsilon_{\sigma,\mu}(\alpha) = \sum_{\alpha'\leq\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha)\sigma(\alpha)(\bot)$. Observe that $\sigma(\alpha)(\bot)$ is a constant with respect to α', and thus can be moved out of the sum, yielding $\epsilon_{\sigma,\mu}(\alpha) = (\sum_{\alpha'\leq\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha))(\sigma(\alpha)(\bot))$.

It suffices to show that $\sum_{\alpha'\leq\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha) = \epsilon_{\sigma,\mu}(C_\alpha) - \mu(C_\alpha - \{\alpha\})$. But this follows immediately from Lemma 2.18. □

Lemma 2.21 *Let σ be a scheduler for PIOA $\mathcal{P}$, and μ be a discrete probability measure on finite execution fragments of $\mathcal{P}$. Let α be a finite execution fragment of $\mathcal{P}$ and a be an action of $\mathcal{P}$ that is enabled in $lstate(\alpha)$. Then*

$$\epsilon_{\sigma,\mu}(C_{\alpha a}) = \mu(C_{\alpha a}) + (\epsilon_{\sigma,\mu}(C_\alpha) - \mu(C_\alpha - \{\alpha\}))\,\sigma(\alpha)(tr_{\alpha,a}).$$

Proof. Observe that $C_{\alpha a} = \cup_q C_{\alpha a q}$. Thus, $\epsilon_{\sigma,\mu}(C_{\alpha a}) = \sum_q \epsilon_{\sigma,\mu}(C_{\alpha a q})$. By Lemma 2.19, the right-hand side is equal to $\sum_q (\mu(C_{\alpha a q}) + (\epsilon_{\sigma,\mu}(C_\alpha) - \mu(C_\alpha - \{\alpha\}))\,\sigma(\alpha)(tr_{\alpha,a})\mu_{\alpha,a}(q))$. Since $\sum_q \mu(C_{\alpha a q}) = \mu(C_{\alpha a})$ and $\sum_q \mu_{\alpha,a}(q) = 1$, this is in turn equal to $\mu(C_{\alpha a}) + (\epsilon_{\sigma,\mu}(C_\alpha) - \mu(C_\alpha - \{\alpha\}))\,\sigma(\alpha)(tr_{\alpha,a})$. Combining the equations yields the result. □

Next, we consider limits of generalized probabilistic execution fragments.

Proposition 2.22 *Let $\epsilon_1, \epsilon_2, \ldots$ be a chain of generalized probabilistic execution fragments of a PIOA $\mathcal{P}$, all generated from the same discrete probability measure μ on finite execution fragments. Then $\lim_{i\to\infty} \epsilon_i$ is a generalized probabilistic execution fragment of $\mathcal{P}$ generated from μ.*

Proof. Let ϵ denote $\lim_{i\to\infty} \epsilon_i$. For each $i \geq 1$, let σ_i be a scheduler such that $\epsilon_i = \epsilon_{\sigma_i,\mu}$, and for each finite execution fragment α, let $p^i_\alpha = \epsilon_{\sigma_i,\mu}(C_\alpha) - \mu(C_\alpha - \{\alpha\})$. For each finite execution α and each action a, let $p^i_{\alpha a} = \epsilon_{\sigma_i,\mu}(C_{\alpha a}) - \mu(C_{\alpha a})$.

By Lemma 2.21, if a is enabled in $lstate(\alpha)$ then $p^i_\alpha \sigma_i(\alpha)(tr_{\alpha,a}) = p^i_{\alpha a}$, and so, if $p^i_{\alpha a} \neq 0$, then $\sigma_i(\alpha)(tr_{\alpha,a}) = p^i_{\alpha a}/p^i_\alpha$.

For each finite execution fragment α, let $p_\alpha = \epsilon(C_\alpha) - \mu(C_\alpha - \{\alpha\})$. For each finite execution fragment α and each action a, let $p_{\alpha a} = \epsilon(C_{\alpha a}) - \mu(C_{\alpha a})$. Define $\sigma(\alpha)(tr_{\alpha,a})$ to be $p_{\alpha a}/p_\alpha$ if $p_\alpha > 0$; otherwise define $\sigma(\alpha)(tr_{\alpha,a}) = 0$. By definition of ϵ and simple manipulations, $\lim_{i\to\infty} p^i_\alpha = p_\alpha$ and $\lim_{i\to\infty} p^i_{\alpha a} = p_{\alpha a}$. It follows that, if $p_\alpha > 0$, then $\sigma(\alpha)(tr_{\alpha,a}) = \lim_{i\to\infty} \sigma_i(\alpha)(tr_{\alpha,a})$.

It remains to show that σ is a scheduler and that $\epsilon_{\sigma,\mu} = \epsilon$. To show that σ is a scheduler, we must show that, for each finite execution fragment α, $\sigma(\alpha)$ is a sub-probability measure. Observe that, for each $i \geq 1$, $\sum_{tr} \sigma_i(\alpha)(tr) = \sum_a \sigma_i(\alpha)(tr_{\alpha a})$. Similarly, $\sum_{tr} \sigma(\alpha)(tr) = \sum_a \sigma(\alpha)(tr_{\alpha a})$. Since each σ_i is a scheduler, it follows that, for each $i \geq 0$, $\sum_a \sigma_i(\alpha)(tr_{\alpha a}) \leq 1$. Thus, also $\lim_{i\to\infty} \sum_a \sigma_i(\alpha)(tr_{\alpha a}) \leq 1$. By interchanging the limit and the sum, we obtain $\sum_a \lim_{i\to\infty} \sigma_i(\alpha)(tr_{\alpha a}) \leq 1$.

We claim that $\sigma(\alpha)(tr_{\alpha,a}) \leq \lim_{i\to\infty} \sigma_i(\alpha)(tr_{\alpha,a})$, which immediately implies that $\sigma(\alpha)(tr_{\alpha a}) \leq 1$, as needed. To see this claim, we consider two cases: If $p_\alpha > 0$, then as shown earlier, $\sigma(\alpha)(tr_{\alpha,a}) = \lim_{i\to\infty} \sigma_i(\alpha)(tr_{\alpha,a})$, which implies the claim. On the other hand, if $p_\alpha = 0$, then $\sigma(\alpha)(tr_{\alpha,a})$ is defined

to be zero, so that $\sigma(\alpha)(tr_{\alpha,a}) = 0$, which is less than or equal to $\lim_{i\to\infty} \sigma_i(\alpha)(tr_{\alpha,a})$, which again implies the claim.

To show that $\epsilon_{\sigma,\mu} = \epsilon$, we show by induction on the length of a finite execution fragment α that $\epsilon_{\sigma,\mu}(C_\alpha) = \epsilon(C_\alpha)$. For the base case, let α consist of a single state q. By Lemma 2.17, $\epsilon_{\sigma,\mu}(C_q) = \mu(C_q)$, and for each $i \geq 1$, $\epsilon_{\sigma_i,\mu}(C_q) = \mu(C_q)$. Thus, $\epsilon(C_q) = \lim_{i\to\infty} \epsilon_{\sigma_i,\mu}(C_q) = \mu(C_q)$, as needed.

For the inductive step, let $\alpha = \tilde{\alpha}aq$. By Lemma 2.19,

$$\lim_{i\to\infty} \epsilon_{\sigma_i,\mu}(C_\alpha) = \lim_{i\to\infty} \left(\mu(C_\alpha) + \left(\epsilon_{\sigma_i,\mu}(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})\right) \sigma_i(\tilde{\alpha})(tr_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q)\right).$$

Observe that the left side is $\epsilon(C_\alpha)$. By algebraic manipulation, the equation above becomes

$$\epsilon(C_\alpha) = \mu(C_\alpha) + \left(\left(\lim_{i\to\infty} \epsilon_{\sigma_i,\mu}(C_{\tilde{\alpha}})\right) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})\right) \left(\lim_{i\to\infty} \sigma_i(\tilde{\alpha})(tr_{\tilde{\alpha},a})\right) \mu_{\tilde{\alpha},a}(q).$$

By definition of ϵ, $\lim_{i\to\infty} \epsilon_{\sigma_i,\mu}(C_{\tilde{\alpha}}) = \epsilon(C_{\tilde{\alpha}})$, and by inductive hypothesis, $\epsilon(C_{\tilde{\alpha}}) = \epsilon_{\sigma,\mu}(C_{\tilde{\alpha}})$. Therefore,

$$\epsilon(C_\alpha) = \mu(C_\alpha) + \left(\epsilon_{\sigma,\mu}(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})\right) \left(\lim_{i\to\infty} \sigma_i(\tilde{\alpha})(tr_{\tilde{\alpha},a})\right) \mu_{\tilde{\alpha},a}(q).$$

Also by Lemma 2.19, we obtain that

$$\epsilon_{\sigma,\mu}(C_\alpha) = \mu(C_\alpha) + \left(\epsilon_{\sigma,\mu}(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})\right) \sigma(\tilde{\alpha})(tr_{\tilde{\alpha},a})\mu_{\alpha,a}(q).$$

We claim that the right-hand sides of the last two equations are equal. To see this, consider two cases. First, if $p_{\tilde{\alpha}} > 0$, then we have already shown that $\lim_{i\to\infty} \sigma_i(\tilde{\alpha})(tr_{\tilde{\alpha},a}) = \sigma(\tilde{\alpha}(tr_{\tilde{\alpha},a}))$. Since these two terms are the only difference between the two expressions, the expressions are equal.

On the other hand, if $p_{\tilde{\alpha}} = 0$, then by definition of $p_{\tilde{\alpha}}$, we get that $\epsilon(C_{\tilde{\alpha}}) = \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$. Then the second terms of the two right-hand sides are both equal to zero, which implies that both expressions are equal to the first term $\mu(C_\alpha)$. Again, the two right-hand sides are equal.

Since the right-hand sides are equal, so are the left-hand sides, that is, $\epsilon_{\sigma,\mu}(C_\alpha) = \epsilon(C_\alpha)$, as needed to complete the inductive hypothesis. □

We denote the set of trace distributions of probabilistic executions of a PIOA $\mathcal{P}$ by $tdists(\mathcal{P})$.

2.2.3 Composition

We define composition for PIOAs:

Definition 2.23 *Two PIOAs $\mathcal{P}_1$ and $\mathcal{P}_2$ are* compatible *if $H_1 \cap A_2 = A_1 \cap H_2 = O_1 \cap O_2 = \emptyset$. The* composition *of two compatible PIOAs $\mathcal{P}_1$ and $\mathcal{P}_2$, denoted by $\mathcal{P}_1 \| \mathcal{P}_2$, is the PIOA $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$ where*

- $Q = Q_1 \times Q_2$,
- $\bar{q} = (\bar{q_1}, \bar{q_2})$,
- $I = (I_1 \cup I_2) - (O_1 \cup O_2)$,
- $O = (O_1 \cup O_2)$,
- $H = (H_1 \cup H_2)$,
- *D is the set of triples $((q_1, q_2), a, \mu_1 \times \mu_2)$ such that for $i \in \{1, 2\}$, if a is an action of $\mathcal{P}_i$, then $(q_i, a, \mu_i) \in D_i$, and if a is not an action of $\mathcal{P}_i$ then $\mu_i = \delta(q_i)$.*

If $q = (q_1, q_2)$ is a state of $\mathcal{P}$ then for $i \in \{1, 2\}$, we write $q\lceil\mathcal{P}_i$ to denote q_i. We extend the definition of composition and the $\lceil$ notation to any finite number of arguments, not just two.

2.2.4 Hiding

We define a hiding operation for PIOAs, which hides output actions.

Definition 2.24 *Let $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$ be a PIOA and $S \subseteq O$. Then hide$(\mathcal{P}, S)$ is the PIOA $\mathcal{P}'$ that is is the same as $\mathcal{P}$ except that $O_{\mathcal{P}'} = O_{\mathcal{P}} - S$ and $H_{\mathcal{P}} = H_{\mathcal{P}} \cup S$.*

3 Task-PIOAs

In this section, we introduce a new "task" mechanism for describing the resolution of nondeterminism. For general PIOAs, we already have a notion of "scheduler", which can use arbitrary knowledge about the past execution in choosing a specific next transition. Such a scheduler is very powerful—too powerful for the security protocol setting. In particular, a scheduler's choice of transition may depend on information that is supposed to be kept secret from the adversarial components. Moreover, the scheduler has very fine-grained control over the precise choice of transition.

To reduce the power of the scheduler, we here define "task-PIOAs", which provide equivalence relations on the actions and on the states of the PIOAs. The action equivalence relation classifies the actions into "tasks", which are units of scheduling.

We begin by defining task-PIOAs, in Section 3.1. Then we define task schedulers, in Section 3.2, which are a variant of our schedulers with coarser granularity (they schedule tasks rather than specific transitions). Section 3.3 defines directly how a task scheduler generates a probability measure on execution fragments, for a closed task-PIOA. Then, in a rather lengthy diversion, it relates this definition to the more traditional definitions for PIOAs, by showing that the resulting probability measure is in fact generated by some traditional scheduler. The next two sections define composition and hiding, for task-PIOAs.

Then, we develop our notions of implementation between task-PIOAs. In Section 3.6, we define the notion of an "environment" for a task-PIOA. We use this, in Section 3.7, to define what it means for one task-PIOA to implement another. Finally, in Section 3.8, we define our new kind of simulation relation between closed task-PIOAs, and prove that it is sound with respect to our implementation notion.

3.1 Task-PIOAs

Definition 3.1 *We define a* task-*PIOA, to be a pair $\mathcal{T} = (\mathcal{P}, R)$, where*

- *$\mathcal{P} = (Q, \bar{q}, I, O, H, D)$ is a PIOA (satisfying next-transition determinism).*
- *R is an equivalence relation on the locally-controlled actions ($O \cup H$).*
  We refer to the equivalence classes of R as tasks*. If a task consists of only output actions or only internal actions we call it, respectively, an* output task *or an* internal task*.*

A task T is enabled *in a state q if there is some action in T that is enabled in q. A task T is* enabled *in a set of states S provided that T is enabled in every $q \in S$.*

We require a task-PIOA to satisfy the following condition:

- **Next-action determinism:** *For every state $q \in Q$ and every task $T \in R$, there is at most one action $a \in T$ that is enabled in q.*

We denote the relation R of a task-PIOA $\mathcal{T}$ by $R_{\mathcal{T}}$.

The non-probabilistic executions and traces of a task-PIOA $\mathcal{T} = (\mathcal{P}, R)$ are defined to be the executions and traces of the underlying PIOA $\mathcal{P}$.

Note that in prelimimary versions of this work [CCK+05], we defined a task-PIOA to have an equivalence relation on states in addition to an equivalence relation on actions. We also had three more axioms, called random-choice consistency, transition consistency, and enabling consistency. These axioms imposed restrictions on the branching capabilities of the modeled components. For example, we

required the states resulting from internal random choices of an adversary to be equivalent; a random choice could not result in enabling a task in one branch and disabling it in another. In this work, we generalize this older definition of a task-PIOA by removing the state relation and the related consistency axioms.

3.2 Task Schedulers

Here we define our notion of a "task scheduler", which chooses the next task to perform. For a closed task-PIOA (that is, one with no input actions), a task scheduler resolves all nondeterminism, because of the *next-action determinism* property of task-PIOAs and the *next-transition determinism* property of general PIOAs.

Our notion of task scheduler is *oblivious*—that is, it is just a sequence of tasks.

Definition 3.2 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed task-PIOA where $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$. A* task scheduler *for $\mathcal{T}$ is defined to be a finite or infinite sequence $\rho = T_1\, T_2\, \ldots$ of tasks in R.*

3.3 Probabilistic executions and trace distributions

Definition 3.3 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA where $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$. The function $apply(,)$ takes a discrete probability measure on finite execution fragments and a task schedule and returns a probability measure on execution fragments. It is defined recursively as follows:*

1. *$apply(\mu, \lambda) = \mu$ (recall that λ is the empty sequence).*
2. *Let T be a single task. Given finite execution fragment α, $apply(\mu, T)(\alpha)$ is defined to be $p_1(\alpha) + p_2(\alpha)$, given as follows.*

$$p_1(\alpha) = \begin{cases} \mu(\alpha')\rho(q) & \text{if } \alpha \text{ can be written as } \alpha'\, a\, q, \text{ with } \alpha' \in supp(\mu), a \in T, \\ & \text{and } (lstate(\alpha'), a, \rho) \in D. \\ 0 & \text{otherwise.} \end{cases}$$

   Notice, in the first case, transition determinism and the definition of execution fragments imply that there is exactly one such ρ, so p_1 is well-defined.

$$p_2(\alpha) = \begin{cases} \mu(\alpha) & \text{if } T \text{ is not enabled in } lstate(\alpha), \\ 0 & \text{otherwise.} \end{cases}$$

3. *If ρ is finite and of the form $\rho' T$, then $apply(\mu, \rho) = apply(apply(\mu, \rho'), T)$.*
4. *If ρ is infinite, let ρ_i denote the length-i prefix of ρ and let ϵ_i be $apply(\mu, \rho_i)$. Then $apply(\mu, \rho) = \lim_{i \to \infty}(\epsilon_i)$.*

Lemma 3.4 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed action-deterministic task-PIOA. Let μ be a discrete probability measure over finite execution fragments of $\mathcal{P}$ and let T be a task. Let p_1 and p_2 be the functions used in the definition of $apply(\mu, T)$. Then:*

1. *for each state q, $p_1(q) = 0$;*
2. *for each finite execution fragment α,*

$$\mu(\alpha) = p_2(\alpha) + \sum_{(a,q):\alpha a q \in \mathsf{Frags}^*(\mathcal{P})} p_1(\alpha a q).$$

Proof. Item (1) follows trivially from the definition of $p_1(q)$.

For Item (2), we observe the following facts.

- If T is not enabled from $lstate(\alpha)$, then, by definition of p_2, $\mu(\alpha) = p_2(\alpha)$. Furthermore, for each action a and each state q such that αaq is an execution fragment, we claim that $p_1(\alpha aq) = 0$. Indeed, if $a \notin T$, then the first case of the definition of $p_1(\alpha)$ trivially does not apply; if $a \in T$, then, since T is not enabled from $lstate(\alpha)$, there is no ρ such that $(lstate(\alpha), a, \rho) \in \mathcal{D}_{\mathcal{P}}$, and thus, again, the first case of the definition of $p_1(\alpha)$ does not apply.
- If T is enabled from $lstate(\alpha)$, then trivially $p_2(\alpha) = 0$. Furthermore, we claim that $\mu(\alpha) = \sum_{(a,q)} p_1(\alpha aq)$. By action determinism, only one action $b \in T$ is enabled from $lstate(\alpha)$. By definition of p_1, $p_1(\alpha aq) = 0$ if $a \neq b$ (either $a \notin T$ or a is not enabled from $lstate(\alpha)$). Thus,

$$\sum_{(a,q)} p_1(\alpha aq) = \sum_{q} p_1(\alpha bq) = \sum_{q} \mu(\alpha)\mu_{\alpha,b}(q).$$

  This in turn is equal to $\mu(\alpha)$ since $\sum_q \mu_{\alpha,b}(q) = 1$.

In each case, we get $\mu(\alpha) = p_2(\alpha) + \sum_{(a,q)} p_1(\alpha aq)$, as needed. □

Lemma 3.5 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed action-deterministic task-PIOA. Let μ be a discrete probability measure over finite execution fragments and ρ be a finite sequence of tasks. Then $apply(\mu, \rho)$ is a discrete probability measure over finite execution fragments.*

Proof. By a simple inductive argument. The key part of the inductive step consists of the claim that, for each measure ϵ on finite executions fragments and each task T, $apply(\epsilon, T)$ is a probability measure over finite execution fragments.

Let ϵ' be $apply(\epsilon, T)$. The fact that ϵ' is a measure on finite execution fragments follows directly by Item (2) of Definition 3.3. To show that ϵ' is in fact a probability measure, we show that $\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon'(\alpha) = 1$. By Item (2) of Definition 3.3,

$$\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon'(\alpha) = \sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} (p_1(\alpha) + p_2(\alpha)).$$

Rearranging terms, we obtain

$$\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon'(\alpha) = \sum_{q} p_1(q) + \sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} (p_2(\alpha) + \sum_{(a,q):\alpha aq \in \mathsf{Frags}^*(\mathcal{P})} p_1(\alpha aq)).$$

By Lemma 3.4, the right side becomes $\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon(\alpha)$, which equals 1 by the inductive hypothesis. Therefore $\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon'(\alpha) = 1$, as needed. □

Lemma 3.6 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed action-deterministic task-PIOA and let T be a task in R. Define $\mu' = apply(\mu, T)$. Then, for each finite execution fragment α:*

1. *If α consists of a single state q, then $\mu'(C_\alpha) = \mu(C_\alpha)$.*
2. *If $\alpha = \tilde{\alpha} aq$ and $a \notin T$, then $\mu'(C_\alpha) = \mu(C_\alpha)$.*
3. *If $\alpha = \tilde{\alpha} aq$ and $a \in T$, then $\mu'(C_\alpha) = \mu(C_\alpha) + \mu(\tilde{\alpha})\mu_{\tilde{\alpha},a}(q)$.*

Proof. Let p_1 and p_2 be the functions used in the definition of $apply(\mu, T)$, and let α be a finite execution fragment. By definition of a cone and of μ', $\mu'(C_\alpha) = \sum_{\alpha' | \alpha \leq \alpha'} (p_1(\alpha') + p_2(\alpha'))$. By definition of a cone and Lemma 3.4, $\mu(C_\alpha) = \sum_{\alpha' | \alpha \leq \alpha'} (p_2(\alpha') + \sum_{(a,q):\alpha' aq \in \mathsf{Frags}^*(\mathcal{P})} p_1(\alpha' aq)) = \sum_{\alpha' | \alpha \leq \alpha'} (p_1(\alpha') + p_2(\alpha')) - p_1(\alpha)$. Thus, $\mu'(C_\alpha) = \mu(C_\alpha) + p_1(\alpha)$. We distinguish three cases. If α consists of a single state, then $p_1(\alpha) = 0$ by Lemma 3.4, yielding $\mu'(C_\alpha) = \mu(C_\alpha)$. If $\alpha = \tilde{\alpha} aq$ and $a \notin T$, then $p_1(\alpha) = 0$ by definition, yielding $\mu'(C_\alpha) = \mu(C_\alpha)$. Finally, if $\alpha = \tilde{\alpha} aq$ and $a \in T$, then $p_1(\alpha) = \mu(\tilde{\alpha})\mu_{\tilde{\alpha},a}(q)$ by definition, yielding $\mu'(C_\alpha) = \mu(C_\alpha) + \mu(\tilde{\alpha})\mu_{\tilde{\alpha},a}(q)$. □

Lemma 3.7 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed action-deterministic task-PIOA. Let μ be a discrete measure over finite execution fragments, T a task, and $\mu' = apply(\mu, T)$. Then $\mu \leq \mu'$.*

Proof. Follows directly by Lemma 3.6. □

Lemma 3.8 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed action-deterministic task-PIOA. Let μ be a discrete measure over finite execution fragments and let ρ_1 and ρ_2 be two finite sequences of tasks such that ρ_1 is a prefix of ρ_2. Then $apply(\mu, \rho_1) \leq apply(\mu, \rho_2)$.*

Proof. Simple inductive argument using Lemma 3.7 for the inductive step. □

Lemma 3.9 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed task-PIOA. Let μ be a discrete measure over finite execution fragments. Then $apply(\mu, \lambda)$ is a generalized probabilistic execution fragment generated by μ.*

Proof. Follows directly from the definitions, by defining a scheduler σ such that $\sigma(\alpha)(\mathsf{tran}) = 0$ for each finite execution fragment α and each transition tran. □

Lemma 3.10 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed action-deterministic task-PIOA. Let μ be a discrete probability measure over finite execution fragments of $\mathcal{P}$, ρ a task scheduler for $\mathcal{T}$, and q a state of $\mathcal{T}$. Then $apply(\mu, \rho)(C_q) = \mu(C_q)$.*

Proof. We prove the result for finite ρ's by induction on the length of ρ. The infinite case then follows immediately. The base case is trivial since, by definition, $apply(\mu, \rho) = \mu$. For the inductive step, let $\rho = \rho' T$, and let ϵ be $apply(\mu, \rho')$. By Definition 3.3, $apply(\mu, \rho) = apply(\epsilon, T)$. By induction, $\epsilon(C_q) = \mu(C_q)$. Therefore it suffices to show $apply(\epsilon, T)(C_q) = \epsilon(C_q)$.

Let ϵ' be $apply(\epsilon, T)$. By definition of cone, $\epsilon'(C_q) = \sum_{\alpha : q \leq \alpha} \epsilon'(\alpha)$. By Lemma 3.5, both ϵ and ϵ' are measures over finite execution fragments; therefore we can restrict the sum to finite execution fragments. Let p_1 and p_2 be the two functions used for the computation of $\epsilon'(\alpha)$ according to Item (2) in Definition 3.3. Then $\epsilon'(C_q) = \sum_{\alpha \in \mathsf{Execs}^*(\mathcal{P}) : q \leq \alpha} (p_1(\alpha) + p_2(\alpha))$. By rearranging terms, we get $\epsilon'(C_q) = p_1(q) + \sum_{\alpha \in \mathsf{Execs}^*(\mathcal{P}) : q \leq \alpha} (p_2(\alpha) + \sum_{(a,s)} p_1(C_{\alpha a s}))$. By Lemma 3.4, the right side of the equation above is $\sum_{\alpha : q \leq \alpha} \epsilon(\alpha)$, which is precisely $\epsilon(C_q)$. □

Lemma 3.11 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed action-deterministic task-PIOA. If ϵ is a generalized probabilistic execution fragment generated by a measure μ, then, for each task T, $apply(\epsilon, T)$ is a generalized probabilistic execution fragment generated by μ.*

Proof. Suppose ϵ is generated by μ together with a scheduler σ (that is, $\epsilon_{\sigma,\mu} = \epsilon$). Let ϵ' be $apply(\epsilon, T)$. Let σ' be a new scheduler such that, for each finite execution fragment α,

- if $\epsilon'(C_\alpha) - \mu(C_\alpha - \{\alpha\}) = 0$, then $\sigma'(\alpha)(\mathsf{tran}) = 0$;
- otherwise,
  - if $\mathsf{tran} \in D(lstate(\alpha))$ and $\mathsf{act}(\mathsf{tran}) \in T$,

$$\sigma'(\alpha)(\mathsf{tran}) = \frac{\epsilon(C_\alpha) - \mu(C_\alpha - \{\alpha\})}{\epsilon'(C_\alpha) - \mu(C_\alpha - \{\alpha\})} (\sigma(\alpha)(\mathsf{tran}) + \sigma(\alpha)(\bot)),$$

  - otherwise,

$$\sigma'(\alpha)(\mathsf{tran}) = \frac{\epsilon(C_\alpha) - \mu(C_\alpha - \{\alpha\})}{\epsilon'(C_\alpha) - \mu(C_\alpha - \{\alpha\})} \sigma(\alpha)(\mathsf{tran}).$$

Here $D(lstate(\alpha))$ denotes the set of transitions of D with source state $lstate(\alpha)$ and $\mathsf{act}(\mathsf{tran})$ denotes the action that occurs in tran. We first prove that σ', thus defined, is a scheduler. We prove by induction on the length of a finite execution fragment α that $\epsilon_{\sigma',\mu}(C_\alpha) = \epsilon'(C_\alpha)$.

For the base case, let $\alpha = q$. By Lemma 2.17, $\epsilon_{\sigma,\mu}(C_q) = \mu(C_q)$ and $\epsilon_{\sigma',\mu}(C_q) = \mu(C_q)$. Thus, $\epsilon_{\sigma',\mu}(C_q) = \epsilon_{\sigma,\mu}(C_q)$. By definition, the right-hand-side is equal to $\epsilon(C_q)$, which is equal to $\epsilon'(C_q)$ by Lemma 3.10. Thus, $\epsilon_{\sigma',\mu}(C_q) = \epsilon'(C_q)$, as needed.

For the inductive step, let $\alpha = \tilde{\alpha}aq$. By Lemma 2.17 and the definition of the measure of a cone (Equation (1)), we get

$$\epsilon_{\sigma',\mu}(C_\alpha) = \mu(C_\alpha) + \sum_{\alpha' \leq \tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma',\alpha'}(C_{\tilde{\alpha}})\mu_{\sigma'(\tilde{\alpha})}(a,q).$$

We know that a is enabled from $lstate(\tilde{\alpha})$, because α is an execution fragment of $\mathcal{P}$. Thus, $\mathsf{tran}_{\tilde{\alpha},a}$ and $\mu_{\tilde{\alpha},a}$ are defined. By expanding $\mu_{\sigma'(\tilde{\alpha})}(a,q)$ in the equation above, we get

$$\epsilon_{\sigma',\mu}(C_\alpha) = \mu(C_\alpha) + \sum_{\alpha' \leq \tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma',\alpha'}(C_{\tilde{\alpha}})\sigma'(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q). \tag{2}$$

We distinguish three cases.

1. $\epsilon'(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) = 0$.

   By inductive hypothesis, $\epsilon_{\sigma',\mu}(C_{\tilde{\alpha}}) = \epsilon'(C_{\tilde{\alpha}})$. Then by Lemma 2.19, $\epsilon_{\sigma',\mu}(C_\alpha) = \mu(C_\alpha)$. It is therefore sufficient to show that $\epsilon'(C_\alpha) = \mu(C_\alpha)$.

   By Lemma 3.7, $\epsilon(C_{\tilde{\alpha}}) \leq \epsilon'(C_{\tilde{\alpha}})$. Thus, using $\epsilon'(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) = 0$, we get $\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) \leq 0$. On the other hand, from Lemma 2.18 and the fact that $\epsilon = \epsilon_{\sigma,\mu}$, we have $\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) \geq 0$. Thus, $\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) = 0$. Now, using Lemma 2.19 and the fact that $\epsilon_{\sigma,\mu} = \epsilon$ and $\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) = 0$, we get $\epsilon(C_\alpha) = \mu(C_\alpha)$.

   Since $C_{\tilde{\alpha}} - \{\tilde{\alpha}\}$ is a union of cones, we may use Lemma 3.7 to obtain $\mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) \leq \epsilon(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$. Adding $\epsilon(\{\tilde{\alpha}\})$ on both sides, we get $\mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) + \epsilon(\{\tilde{\alpha}\}) \leq \epsilon(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) + \epsilon(\{\tilde{\alpha}\}) = \epsilon(C_{\tilde{\alpha}})$. Since $\epsilon(C_{\tilde{\alpha}}) = \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$, the previous inequalities imply $\epsilon(C_{\tilde{\alpha}}) + \epsilon(\{\tilde{\alpha}\}) \leq \epsilon(C_{\tilde{\alpha}})$, therefore $\epsilon(\{\tilde{\alpha}\}) = 0$. By Lemma 3.6 (Items (2) and (3)), we have $\epsilon'(C_\alpha) = \epsilon(C_\alpha) = \mu(C_\alpha)$, as needed.

2. $\epsilon'(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) > 0$ and $a \notin T$.

   By Equation (2) and the definition of σ', we know that $\epsilon_{\sigma',\mu}(C_\alpha)$ equals

   $$\mu(C_\alpha) + \sum_{\alpha' \leq \tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma',\alpha'}(C_{\tilde{\alpha}})\frac{\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})}{\epsilon'(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})}\sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q).$$

   Observe that in the sum above only the factors $\mu(\alpha')\epsilon_{\sigma',\alpha'}(C_{\tilde{\alpha}})$ are not constant with respect to the choice of α'. By Lemma 2.18 and algebraic manipulation, $\sum_{\alpha' \leq \tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma',\alpha'}(C_{\tilde{\alpha}}) = \epsilon_{\sigma',\mu}(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$. By inductive hypothesis, $\epsilon_{\sigma',\mu}(C_{\tilde{\alpha}}) = \epsilon'(C_{\tilde{\alpha}})$. Thus, replacing $\sum_{\alpha' \leq \tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma',\alpha'}(C_{\tilde{\alpha}})$ with $\epsilon'(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$ and simplifying the resulting expression, we get

   $$\epsilon_{\sigma',\mu}(C_\alpha) = \mu(C_\alpha) + (\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\,\sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q).$$

   By definition, $\epsilon = \epsilon_{\sigma,\mu}$. Therefore, by Lemma 2.19, the right side of the equation above is $\epsilon(C_\alpha)$. Moreover, $\epsilon(C_\alpha) = \epsilon'(C_\alpha)$ by Lemma 3.6, Item (2). Thus, $\epsilon_{\sigma',\mu}(C_\alpha) = \epsilon'(C_\alpha)$, as needed.

3. $\epsilon'(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) > 0$ and $a \in T$.

   As in the previous case, $\epsilon_{\sigma',\mu}(C_\alpha)$ equals

   $$\mu(C_\alpha) + (\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))(\sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a}) + \sigma(\tilde{\alpha})(\bot))\mu_{\tilde{\alpha},a}(q).$$

   Also shown in the previous case, we have

   $$\epsilon(C_\alpha) = \mu(C_\alpha) + (\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\,\sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q).$$

Therefore,

$$\epsilon_{\sigma',\mu}(C_\alpha) = \epsilon(C_\alpha) + (\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\sigma(\tilde{\alpha})(\bot)\mu_{\tilde{\alpha},a}(q).$$

By definition, $\epsilon = \epsilon_{\sigma,\mu}$. Using Lemma 2.20, we may substitute $\epsilon(\tilde{\alpha})$ for $(\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\sigma(\tilde{\alpha})(\bot)$. Now we have

$$\epsilon_{\sigma',\mu}(C_\alpha) = \epsilon(C_\alpha) + \epsilon(\tilde{\alpha})\mu_{\tilde{\alpha},a}(q).$$

The desired result now follows from Lemma 3.6, Item (3).

□

Lemma 3.12 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed action-deterministic task-PIOA. For each probability measure μ on finite execution fragments and each finite sequence of tasks ρ, $apply(\mu, \rho)$ is a generalized probabilistic execution fragment generated by μ.*

Proof. Simple inductive argument using Lemma 3.9 for the base case and Lemma 3.11 for the inductive step. □

Lemma 3.13 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed action-deterministic task-PIOA. For each measure μ on finite execution fragments and each infinite sequence of tasks ρ, $apply(\mu, \rho)$ is a generalized probabilistic execution fragment generated by μ.*

Proof. For each $i \geq 0$, let ρ_i denote the length-i prefix of ρ and let ϵ_i be $apply(\mu, \rho_i)$. By Lemmas 3.12 and 3.8, the sequence $\epsilon_0, \epsilon_1, \ldots$ is a chain of generalized probabilistic execution fragments generated by μ. By Proposition 2.22, $\lim_{i\to\infty} \epsilon_i$ is a generalized probabilistic execution fragment generated by μ. This suffices, since $apply(\mu, \rho)$ is $\lim_{i\to\infty} \epsilon_i$ by definition. □

Now we can prove Proposition 3.14, our main target. It says that, for any μ and ρ, the probability measure on execution fragments generated by $apply(\mu, \rho)$ is "standard", in that it can be obtained from μ and a scheduler in the sense of Definition 2.15.

Proposition 3.14 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed action-deterministic task-PIOA. For each measure μ on finite execution fragments and each sequence of tasks ρ, $apply(\mu, \rho)$ is a generalized probabilistic execution fragment generated by μ.*

Proof. Follows directly by Lemmas 3.12 and 3.13. □

Lemma 3.15 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed action-deterministic task-PIOA. Let $\rho_1, \rho_2, \cdots$ be a finite or infinite sequence of finite task schedulers and let μ be a discrete probability measure on finite execution fragments. For each $i > 0$, let $\epsilon_i = apply(\mu, \rho_1\rho_2\cdots\rho_i)$, where $\rho_1\cdots\rho_i$ denotes the concatenation of the sequences ρ_1 through ρ_i. Let ρ be the concatenation of all the ρ_i's, and let $\epsilon = apply(\mu, \rho)$. Then the ϵ_i's form a chain and $\epsilon = \lim_{i\to\infty} \epsilon_i$.*

Proof. The fact that the ϵ_i's form a chain follows from Lemma 3.7. For the limit property, if the sequence $\rho_1, \rho_2, \ldots$ is finite, then the result is immediate. Otherwise, simply observe that the sequence $\epsilon_1, \epsilon_2, \ldots$ is a sub-sequence of the sequence used in the definition of $apply(\mu, \rho_1\rho_2\ldots)$, therefore they have the same limit. □

A *generalized probabilistic execution fragment* of a closed task-PIOA $\mathcal{T}$ is any generalized probabilistic execution fragment of the underlying PIOA $\mathcal{P}$ that is generated from any μ and any task scheduler ρ, as $apply(\mu, \rho)$. If $supp(\mu)$ is included in the set of states of $\mathcal{P}$, then we call $apply(\mu, \rho)$ a *probabilistic execution fragment* of $\mathcal{T}$. Finally, for the start state $\bar{q}$, we call $apply(\bar{q}, \rho)$ a *probabilistic execution* of $\mathcal{T}$.

Now we consider trace distributions of task-PIOAs. Namely, for any μ and ρ, we write $tdist(\mu, \rho)$ as shorthand for $tdist(apply(\mu, \rho))$. We write $tdist(\rho)$ as shorthand for $tdist(\delta(apply(\bar{q},))\rho)$, where $\bar{q}$ is the unique start state. A *trace distribution* of $\mathcal{T}$ is any $tdist(\rho)$. We use $tdists(\mathcal{T})$ for a closed task-PIOA T to denote the set $\{tdist(\rho) : \rho \text{ is a task scheduler for } \mathcal{T}\}$.

3.4 Composition

The systems in this paper are described as compositions of task-PIOAs. Here we show how to regard such a composition as a task-PIOA.

Definition 3.16 *Two task-PIOAs $\mathcal{T}_1 = (\mathcal{P}_1, R_1)$ and $\mathcal{T}_2 = (\mathcal{P}_2, R_2)$ are said to be* compatible *provided that the underlying PIOAs $\mathcal{P}_1$ and $\mathcal{P}_2$ are compatible. In this case, we define their* composition $\mathcal{T}_1 \| \mathcal{T}_2$ *to be the task-PIOA $(\mathcal{P}_1 \| \mathcal{P}_2, R_1 \cup R_2)$.*

Lemma 3.17 *If $\mathcal{T}_1$ and $\mathcal{T}_2$ are compatible action-deterministic task-PIOAs, then $\mathcal{T}_1 \| \mathcal{T}_2$ is also action-deterministic.*

3.5 Hiding

We define a hiding operation for task-PIOAs, which hides output tasks.

Definition 3.18 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a task-PIOA where $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$, and let $\mathcal{U} \subseteq R$ be a set of output tasks. Let $S = \cup_{T \in \mathcal{U}} T$, that is, S is the set of actions in all the tasks in $\mathcal{U}$. Then we define $hide(\mathcal{T}, \mathcal{U})$ to be $(hide(\mathcal{P}, S), R)$.*

3.6 Environments

We define the notion of environment as follows.

Definition 3.19 *Suppose $\mathcal{T}$ and $\mathcal{E}$ are task-PIOAs. We say that $\mathcal{E}$ is an* environment *for $\mathcal{T}$ iff $\mathcal{E}$ is compatible with $\mathcal{T}$, $\mathcal{T} \| \mathcal{E}$ is closed and $\mathcal{E}$ has a special output action named accept.*

The special *accept* output action is used by the environment to distinguish between different task-PIOAs.

3.7 Implementation

Our notion of implementation for task-PIOAs is based on probabilistic executions that look the same to *any* environment for the PIOAs. This notion of implementation makes sense only for comparable task-PIOAs.

Definition 3.20 *Two task-PIOAs $(\mathcal{P}_1, R_1)$ and $(\mathcal{P}_2, R_2)$ are* comparable *if $\mathcal{P}_1$ and $\mathcal{P}_2$ are comparable (have the same external signature) .*

We now define the $\leq_0$-implementation notion for task-PIOAs.

Definition 3.21 *Suppose $\mathcal{T}_1$ and $\mathcal{T}_2$ are two comparable task-PIOAs. We say that $\mathcal{T}_1 \leq_0 \mathcal{T}_2$ provided that, for every environment $\mathcal{E}$ for both $\mathcal{T}_1$ and $\mathcal{T}_2$, $tdists(\mathcal{T}_1 || \mathcal{E}) \subseteq tdists(\mathcal{T}_2 || \mathcal{E})$.*

3.8 Simulation relations

We first give some auxiliary definitions that are used in the definitions of a simulation relation.

The definition below formulates a "consistency" condition between a distribution over executions and a task sequence. A discrete distribution over executions is said to be consistent with a task sequence if each execution in the support of that distribution can be obtained by performing the tasks in the sequence (in the order of their occurrence). We use this condition in order to avoid useless proof obligations in our definition of simulation relations.

Definition 3.22 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed task-PIOA and ϵ be a discrete distribution over finite executions of $\mathcal{P}$, and ρ a finite task sequence for $\mathcal{T}$. Then we say that ϵ is* consistent with *ρ provided that the following holds: For every $\alpha \in supp(\epsilon), \alpha \in supp(apply(\delta(fstate(\alpha)), \rho))$. That is, each execution in the support of ϵ is "consistent with" the task scheduler ρ.*

We now define the operation $full$ that will subsequently be used in the definition of a simulation relation. Suppose that we have a mapping, which, given a task sequence ρ and a task T of an automaton $\mathcal{T}_1$, yields a task sequence of another automaton $\mathcal{T}_2$. The intuition is that, for each task T of $\mathcal{T}_1$, this mapping gives a task sequence of $\mathcal{T}_2$ that corresponds to the task T. The operation $full$ transforms this mapping to a mapping, which given a task sequence ρ of T_1 gives a "full" task sequence of $\mathcal{T}_2$ that corresponds to ρ, not just the incremental part that corresponds to a single task of $\mathcal{T}_1$.

Definition 3.23 *If $\mathcal{T}_1 = (\mathcal{P}_1, R_1)$ and $\mathcal{T}_2 = (\mathcal{P}_2, R_2)$ are two task-PIOAs, and if $c : ({R_1}^* \times R_1) \rightarrow {R_2}^*$, then define $full(c) : {R_1}^* \rightarrow {R_2}^*$ recursively, as follows:*

$$full(c)(\lambda) = \lambda$$
$$full(c)(\rho\, T) = full(c)(\rho)\, c(\rho, T).$$

We now define what it means for a relation to be a simulation relation from one task-PIOA to another. The definition of a simulation relation is new and we wish to establish its soundness with respect to the $\leq_0$ relation.

Definition 3.24 *Let $\mathcal{T}_1 = (\mathcal{P}_1, R_1)$ and $\mathcal{T}_2 = (\mathcal{P}_2, R_2)$ be two comparable closed action-deterministic task-PIOAs. Let R be a relation from discrete distributions over finite executions of $\mathcal{P}_1$ to discrete distributions over finite executions of $\mathcal{P}_2$, satisfying the condition: If $\epsilon_1\ R\ \epsilon_2$ then $tdist(\epsilon_1) = tdist(\epsilon_2)$. Let $c : ({R_1}^* \times R_1) \rightarrow {R_2}^*$. Then we say that R is a* simulation relation *from $\mathcal{T}_1$ to $\mathcal{T}_2$ using c provided that the following properties hold:*

1. ***Start condition:*** *$\delta(\bar{q}_1)\ R\ \delta(\bar{q}_2)$.*
2. ***Step condition:*** *If $\epsilon_1\ R\ \epsilon_2$, $\rho_1 \in {R_1}^*$, ϵ_1 is consistent with ρ_1, ϵ_2 is consistent with $full(c)(\rho_1)$, and $T \in R_1$, then $\epsilon_1'\ \mathcal{E}(R)\ \epsilon_2'$ where: $\epsilon_1' = apply(\epsilon_1, T)$ and $\epsilon_2' = apply(\epsilon_2, c(\rho_1, T))$.*

We say that R is a simulation relation *from $\mathcal{T}_1$ to $\mathcal{T}_2$ provided that R is a simulation relation from $\mathcal{T}_1$ to $\mathcal{T}_2$ using c, for some c.*

Lemma 3.25 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two comparable closed action-deterministic task-PIOAs, R a simulation from $\mathcal{T}_1$ to $\mathcal{T}_2$. Let ϵ_1 and ϵ_2 be discrete distributions on finite executions of $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively, such that $\epsilon_1\ \mathcal{E}(R)\ \epsilon_2$. Then $tdist(\epsilon_1) = tdist(\epsilon_2)$.*

Lemma 3.26 *Let $\{\mu_i\}_i$ be a countable family of discrete probability measures on finite execution fragments and let $\{p_i\}_i$ be a countable family of probabilities such that $\sum_i p_i = 1$. Let T be a task. Then, $apply(\sum_i p_i\mu_i, T) = \sum_i p_i\ apply(\mu_i, T)$.*

Proof. Let p_1 and p_2 be the functions used in the definition of $apply(\sum_i p_i\mu_i, T)$, and let, for each i, p_1^i and p_2^i be the functions used in the definition of $apply(\mu_i, T)$. Let α be a finite execution fragment. We show that $p_1(\alpha) = \sum_i p_i p_1^i(\alpha)$ and $p_2(\alpha) = \sum_i p_i p_2^i(\alpha)$. Then it follows that $apply(\sum_i p_i\mu_i, T)(\alpha) = \sum_i p_i\ apply(\mu_i, T)(\alpha)$ since $apply(\sum_i p_i\mu_i, T)(\alpha)$ is defined to be $p_1(\alpha) + p_2(\alpha)$, and $\sum_i p_i\ apply(\mu_i, T)(\alpha) = \sum_i p_i(p_1^i(\alpha) + p_2^i(\alpha)) = \sum_i p_i p_1^i(\alpha) + \sum_i p_i p_2^i(\alpha) = p_1(\alpha) + p_2(\alpha)$.

To prove our claim about p_1 we distinguish two cases. If α can be written as $\alpha'\ a\ q$, where $\alpha' \in supp(\mu), a \in T$, and $(lstate(\alpha'), a, \rho) \in D_{\mathcal{P}}$, then, by Definition 3.3, $p_1(\alpha) = (\sum_i p_i\mu_i)(\alpha')\rho(q)$, and, for each i, $p_1^i(\alpha) = \mu_i(\alpha')\rho(q)$. Thus, $p_1(\alpha) = \sum_i p_i p_1^i(\alpha)$ trivially. Otherwise, again by Definition 3.3, $p_1(\alpha) = 0$, and, for each i, $p_1^i(\alpha) = 0$. Thus, $p_1(\alpha) = \sum_i p_i p_1^i(\alpha)$ trivially.

To prove our claim about p_2 we also distinguish two cases. If T is not enabled in$lstate(\alpha)$, then, by Definition 3.3, $p_2(\alpha) = (\sum_i p_i\mu_i)(\alpha)$, and, for each i, $p_2^i(\alpha) = \mu_i(\alpha)$. Thus, $p_2(\alpha) = \sum_i p_i p_2^i(\alpha)$ trivially. Otherwise, again by Definition 3.3, $p_2(\alpha) = 0$, and, for each i, $p_2^i(\alpha) = 0$. Thus, $p_2(\alpha) = \sum_i p_i p_2^i(\alpha)$ trivially. □

Proposition 3.27 *Let $\{\mu_i\}_i$ be a countable family of discrete probability measures on finite execution fragments and let $\{p_i\}_i$ be a countable family of probabilities such that $\sum_i p_i = 1$. Let ρ be a finite sequence of tasks. Then, $apply(\sum_i p_i\mu_i, \rho) = \sum_i p_i\ apply(\mu_i, \rho)$.*

Proof. We proceed by induction on the length of ρ. If $\rho = \lambda$, then the result is trivial since $apply(\cdot, \lambda)$ is defined to be the identity function, which distributes over convex combinations of probability measures. For the inductive step, let ρ be $\rho' T$. By Definition 3.3, $apply(\sum_i p_i \mu_i, \rho' T) = apply(apply(\sum_i p_i \mu_i, \rho'), T)$. By induction, $apply(\sum_i p_i \mu_i, \rho') = \sum_i p_i\, apply(\mu_i, \rho')$. Thus, we obtain $apply(\sum_i p_i \mu_i, \rho' T) = apply(\sum_i p_i\, apply(\mu_i, \rho'), T)$. By Lemma 3.5, for each i, $apply(\mu_i, \rho')$ is a discrete probability measure over finite execution fragments. By Lemma 3.26, $apply(\sum_i p_i\, apply(\mu_i, \rho'), T) = \sum_i p_i\, apply(apply(\mu_i, \rho'), T)$, and by Definition 3.3, for each i, $apply(apply(\mu_i, \rho'), T) = apply(\mu_i, \rho' T)$. Thus, $apply(\sum_i p_i \mu_i, \rho' T) = \sum_i p_i\, apply(\mu_i, \rho' T)$ as needed. □

Lemma 3.28 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two comparable closed task-PIOAs, let R be a simulation relation from $\mathcal{T}_1$ to $\mathcal{T}_2$, and let corrtasks be a mapping that satisfies the conditions required for a simulation relation. Let ρ_1 and ρ_2 be finite task schedulers of $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively, such that $\rho_2 = full(corrtasks)(\rho_1)$. Let $\epsilon_1 = apply(\delta(\bar{q}_1), \rho_1)$ and $\epsilon_2 = apply(\delta(\bar{q}_2), \rho_2)$ be the respective discrete distributions on finite executions of $\mathcal{T}_1$ and $\mathcal{T}_2$ generated by ρ_1 and ρ_2. Suppose that $\epsilon_1\ \mathcal{E}(R)\ \epsilon_2$.*

Let T be a task of $\mathcal{T}_1$. Let $\epsilon_1' = apply(\delta(\bar{q}_1), \rho_1 T)$ and let $\epsilon_2' = apply(\delta(\bar{q}_2), \rho_2\, corrtasks(\rho_1, T))$. Then $\epsilon_1'\ \mathcal{E}(R)\ \epsilon_2'$.

Proof. Let η_1, η_2 and w be the measures and weighting function that witness $\epsilon_1\ \mathcal{E}(R)\ \epsilon_2$. Observe that $\epsilon_1' = apply(\epsilon_1, T)$ and $\epsilon_2' = apply(\epsilon_2, corrtasks(\rho_1, T))$.

We apply Lemma 2.4: Define the function f on discrete distributions on finite execution fragments of $\mathcal{T}_1$ by $f(\epsilon) = apply(\epsilon, T)$, and the function g on discrete distributions on finite execution fragments of $\mathcal{T}_2$ by $g(\epsilon) = apply(\epsilon, corrtasks(\rho_1, T))$. We show that the hypothesis of Lemma 2.4 is satisfied, which implies that, by Lemma 2.4, $\epsilon_1'\ \mathcal{E}(R)\ \epsilon_2'$, as needed.

Distributivity of f and g follows directly by Proposition 3.27. Let μ_1, μ_2 be two measures such that $w(\mu_1, \mu_2) > 0$. We must show that $f(\mu_1)\ \mathcal{E}(R)\ g(\mu_2)$. Since w is a weighting function for $\epsilon_1\ \mathcal{E}(R)\ \epsilon_2$, $\mu_1\ R\ \mu_2$. Observe that $supp(\mu_1) \subseteq supp(\epsilon_1)$ and $supp(\mu_2) \subseteq supp(\epsilon_2)$; thus, μ_1 is consistent with ρ_1 and μ_2 is consistent with ρ_2. By the step condition for R, $apply(\mu_1, T)\ \mathcal{E}(R)\ apply(\mu_2, corrtasks(\rho_1, T))$. Observe that $apply(\mu_1, T) = f(\mu_1)$ and that $apply(\mu_2, corrtasks(\rho_1, T)) = g(\mu_2)$. Thus, $f(\mu_1)\ \mathcal{E}(R)\ g(\mu_2)$, as needed. □

The main theorem we would like to prove is the following:

Theorem 3.29 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two comparable closed action-deterministic task-PIOAs. If there exists a simulation relation from $\mathcal{T}_1$ to $\mathcal{T}_2$, then $tdists(\mathcal{T}_1) \subseteq tdists(\mathcal{T}_2)$.*

Proof. Let R be the assumed simulation relation from $\mathcal{T}_1$ to $\mathcal{T}_2$. Let ϵ_1 be the probabilistic execution of $\mathcal{T}_1$ generated by $\bar{q}_1$ and a (finite or infinite) task schedule, $T_1, T_2, \cdots$. For each $i > 0$, define ρ_i to be $corrtasks(T_1 \cdots T_{i-1}, T_i)$. Let ϵ_2 be the probabilistic execution generated by $\bar{q}_2$ and the concatenation $\rho_1 \rho_2 \cdots$. We claim that $tdist(\epsilon_1) = tdist(\epsilon_2)$, which suffices.

For each $j \geq 0$, let $\epsilon_{1,j} = apply(\bar{q}_1, T_1 \cdots T_j)$, and $\epsilon_{2,j} = apply(\bar{q}_2, \rho_1 \cdots \rho_j)$. By Lemma 3.15, for each $j \geq 0$, $\epsilon_{1,j} \leq \epsilon_{1,j+1}$ and $\epsilon_{2,j} \leq \epsilon_{2,j+1}$, and furthermore, $\lim_{j\to\infty} \epsilon_{1,j} = \epsilon_1$ and $\lim_{j\to\infty} \epsilon_{2,j} = \epsilon_2$. Also, note that for every $j \geq 0$, $apply(\epsilon_{1,j}, T_{j+1}) = \epsilon_{1,j+1}$ and $apply(\epsilon_{2,j}, \rho_{j+1}) = \epsilon_{2,j+1}$.

Observe that $\epsilon_{1,0} = \delta(\bar{q}_1)$ and $\epsilon_{2,0} = \delta(\bar{q}_2)$. By the start condition for a simulation relation and a trivial expansion, we see that $\epsilon_{1,0}\ \mathcal{E}(R)\ \epsilon_{2,0}$. Then by induction, using Lemma 3.28 for the inductive step, for each $j \geq 0$, $\epsilon_{1,j}\ \mathcal{E}(R)\ \epsilon_{2,j}$. Then, by Lemma 3.25, for each $j \geq 0$, $tdist(\epsilon_{1,j}) = tdist(\epsilon_{2,j})$. By Lemma 2.14, $tdist(\epsilon_1) = \lim_{j\to\infty} tdist(\epsilon_{1,j})$, and $tdist(\epsilon_2) = \lim_{j\to\infty} tdist(\epsilon_{2,j})$. Since for each $j \geq 0$, $tdist(\epsilon_{1,j}) = tdist(\epsilon_{2,j})$, we conclude $tdist(\epsilon_1) = tdist(\epsilon_2)$, as needed. □

In order to use our implementation results in a setting involving polynomial time bounds, we need a slight variant of Theorem 3.29. This variant assumes a constant bound on the lengths of the *corrtasks* sequences, and guarantees a bound on the ratio of the sizes of the high-level and low-level task schedulers.

Theorem 3.30 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two closed task-PIOAs, and $c \in \mathsf{N}$. Suppose there exists a simulation relation from $\mathcal{T}_1$ to $\mathcal{T}_2$ using corrtasks, for which $|corrtasks(\rho, T)| \leq c$ for every ρ and T.*
If τ is a trace distribution of $\mathcal{T}_1$ that is generated by a task scheduler ρ_1, then τ is also generated by some task scheduler ρ_2 for $\mathcal{T}_2$, with $|\rho_2| \leq c|\rho_1|$.

Proof. By examination of the proof of the proof of Theorem 3.29. □

The lemma below captures a special case of the simulation relation definition we have given above. Any relation that satisfies the hypotheses of this lemma is a simulation relation. We use this special case in proving the correctness of the OT protocol.

Lemma 3.31 *Let $\mathcal{T}_1 = (\mathcal{P}_1, R_1)$ and $\mathcal{T}_2 = (\mathcal{P}_2, R_2)$ be two comparable closed action-deterministic task-PIOAs. Let R be a relation from discrete distributions over finite executions of $\mathcal{P}_1$ to discrete distributions over finite executions of $\mathcal{P}_2$, satisfying: If $\epsilon_1\ R\ \epsilon_2$ then $tdist(\epsilon_1) = tdist(\epsilon_2)$. Let $c : ({R_1}^* \times R_1) \to {R_2}^*$. Suppose further that the following conditions hold:*

1. ***Start condition:*** *$\delta(\bar{q}_1)\ R\ \delta(\bar{q}_2)$.*
2. ***Step condition:*** *If $\epsilon_1\ R\ \epsilon_2$, $\rho_1 \in {R_1}^*$, ϵ_1 is consistent with ρ_1, ϵ_2 is consistent with $full(c)(\rho_1)$, and $T \in R_1$, then there exist*
   - *a probability measure p on a countable index set I,*
   - *probability measures ϵ'_{1j}, $j \in I$, on finite executions of $\mathcal{P}_1$, and*
   - *probability measures ϵ'_{2j}, $j \in I$, on finite executions of $\mathcal{P}_2$,*

   such that:
   - *for each $j \in I$, $\epsilon'_{1j}\ R\ \epsilon'_{2j}$,*
   - *$\sum_{j \in I} p(j)(\epsilon'_{1j}) = apply(\epsilon_1, T)$, and*
   - *$\sum_{j \in I} p(j)(\epsilon'_{2j}) = apply(\epsilon_2, c(\rho_1, T))$.*

Then R is a simulation relation from $\mathcal{T}_1$ to $\mathcal{T}_2$ using c.

Proof. The proof is straightforward, using Lemma 2.3. □

Now we give a corollary of the main soundness result, for not-necessarily-closed task-PIOAs.

Corollary 3.32 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two comparable action-deterministic task-PIOAs. Suppose that, for every environment $\mathcal{E}$ for both $\mathcal{T}_1$ and $\mathcal{T}_2$, there exists a simulation relation R from $\mathcal{T}_1 \| \mathcal{E}$ to $\mathcal{T}_2 \| \mathcal{E}$.*
Then $\mathcal{T}_1 \leq_0 \mathcal{T}_2$.

4 Time-Bounded Task-PIOAs

In this section, we impose time bounds on task-PIOAs. We will use this in the next section to define polynomial-time-bounded task-PIOAs.

4.1 Time-Bounded Task-PIOAs

We assume a standard bit-string representation scheme for actions and tasks, which is the same for all task-PIOAs that have these actions and tasks. We write $\langle a \rangle$ for the representation of action a, and $\langle T \rangle$ for the representation of task T.

Definition 4.1 *Task-PIOA $\mathcal{T}$ is said to be* b-time-bounded, *where $b \in \mathsf{R}^{\geq 0}$, provided that:*

1. ***Automaton parts:*** *Every state q, transition tr, and task T has a bit-string representation, which we denote by $\langle q \rangle$, $\langle tr \rangle$, and $\langle T \rangle$, respectively. The length of the bit-string representation of every action, state, transition and task of $\mathcal{T}$ is at most b.*

2. ***Decoding:*** *There is a deterministic Turing machine that, given the representation of a candidate state q, decides whether q is a state of $\mathcal{T}$, and always runs in time at most b. Also, there is a deterministic Turing machine that, given the representation of a candidate state q, decides whether q is the unique start state of $\mathcal{T}$. Similarly for a candidate input action, output action, internal action, transition, output task or internal task. Also, there is a deterministic Turing machine that, given the representation of two candidate actions a_1 and a_2, decides whether $(a_1, a_2) \in R_{\mathcal{T}}$, and always runs in time at most b. Also, there is a deterministic Turing machine that, given the representation of an action a of $\mathcal{T}$ and a task T, decides whether $a \in T$; again, this machine runs in time b.*

3. ***Determining the next action:*** *There is a deterministic Turing machine M_{act} that, given the representation of a state q of $\mathcal{T}$ and the representation of an output or internal task T of $\mathcal{T}$, produces the representation of the unique action a in task T that is enabled in q if one exists, and otherwise produces a special "no-action" indicator. Moreover, M_{act} always runs in time at most b.*

4. ***Determining the next state:*** *There is a probabilistic Turing machine M_{state} that, given the representation of a state q of $\mathcal{T}$, and the representation of an action a of $\mathcal{T}$ that is enabled in q, produces the representation of the next state resulting from the unique transition of $\mathcal{T}$ of the form (q, a, μ). Moreover, M_{state} always runs in time at most b.*

Moreover, we require that every Turing machine mentioned in this definition can be described using a bit string of length at most b, according to some standard encoding of Turing machines.

In the rest of this paper, we will not explicitly distinguish $\langle x \rangle$ from x.

4.2 Composition

We have already defined composition for task-PIOAs. Now we show that the composition of two time-bounded task-PIOAs is also time-bounded, with a bound that is simply related to the bounds for the two components.

Lemma 4.2 *There exists a constant c such that the following holds. Suppose $\mathcal{T}_1$ is a b_1-time-bounded task-PIOA and $\mathcal{T}_2$ is a b_2-time-bounded task-PIOA, where $b_1, b_2 \geq 1$. Then $\mathcal{T}_1 \| \mathcal{T}_2$ is a $c(b_1 + b_2)$-bounded task-PIOA.*

Proof. We describe how the different bounds of Def. 4.1 combine when we compose $\mathcal{T}_1$ and $\mathcal{T}_2$.

1. **Automaton parts:** Every action or task of $\mathcal{T}_1 \| \mathcal{T}_2$ has a standard representation, which is the same as its representation in $\mathcal{T}_1$ or $\mathcal{T}_2$. The length of this representation is, therefore, at most $\max(b_1, b_2)$.

   Every state of $\mathcal{T}_1 \| \mathcal{T}_2$ can be represented with a $2(b_1 + b_2) + 2 \leq 3(b_1 + b_2)$-bit string, by following each bit of the bit-string representations of the states of $\mathcal{T}_1$ and $\mathcal{T}_2$ with a zero, and then concatenating the results, separating them with the string 11. Likewise, every transition of $\mathcal{T}_1 \| \mathcal{T}_2$ can be represented as a $3(b_1 + b_2)$-bit string, by combining the representations of transitions of one or both of the component automata.

2. **Decoding:** It is possible to decide whether a candidate state $q = (q_1, q_2)$ is a state of $\mathcal{T}_1 \| T_2$ by checking if q_1 is a state of $\mathcal{T}_1$ and q_2 is a state of $\mathcal{T}_2$. Similar verifications can be carried out for candidate start states.

It is possible to decide if a candidate input action is an input action of $\mathcal{T}_1 \| T_2$ by checking if it is an input action of T_1 or $\mathcal{T}_2$ but not an output action of $\mathcal{T}_1$ or $\mathcal{T}_2$. It is possible to decide if a candidate internal (resp. output) action is an internal (resp. output) action of $\mathcal{T}_1 \| T_2$ by checking if it is an internal (resp. output) action of $\mathcal{T}_1$ or $\mathcal{T}_2$. A similar verification can be carried out for internal and output tasks.

Given two candidate actions a_1 and a_2 of $\mathcal{T}_1 \| \mathcal{T}_2$, it is possible to decide whether $(a_1, a_2) \in R_{\mathcal{T}_1 \| \mathcal{T}_2}$ by checking if $(a_1, a_2) \in R_{\mathcal{T}_1}$ or $(a_1, a_2) \in R_{\mathcal{T}_2}$. Also, given an action a of $\mathcal{T}_1 \| \mathcal{T}_2$ and a task T of $\mathcal{T}_1 \| \mathcal{T}_2$, it is possible to decide whether $a \in T$ by determining a component automaton $\mathcal{T}_i$ that has T as a task and using the procedure assumed for $\mathcal{T}_i$ to check whether $a \in \mathcal{T}$.

All these verifications can be done in time $\mathcal{O}(b_1 + b_2)$.

3. **Determining the next action:** Assume M_{act1} and M_{act2} are the deterministic Turing machines described in Part 3 of Def. 4.1 for $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively. We define M_{act} for $\mathcal{T}_1 \| \mathcal{T}_2$ as the deterministic Turing machine that, given state $q = (q_1, q_2)$ of $\mathcal{T}_1 \| \mathcal{T}_2$ where $q_1 = q \lceil \mathcal{T}_1$ and $q_2 = q \lceil \mathcal{T}_2$ and task T, outputs:

   - The action (or "no-action" indicator) that is output by $M_{act1}(q_1, T)$, if T is an output or internal task of $\mathcal{T}_1$.
   - The action (or "no-action" indicator) that is output by $M_{act2}(q_2, T)$ if T is an output or internal task of $\mathcal{T}_2$.

   M_{act} always operates within time $\mathcal{O}(b_1 + b_2)$: this time is sufficient to determine whether T is an output or internal task of $\mathcal{T}_1$ or $\mathcal{T}_2$, to extract the needed part of q to supply to M_{act1} or M_{act2}, and to run M_{act1} or M_{act2}.

4. **Determining the next state:** Assume M_{state1} and M_{state2} are the probabilistic Turing machines described in Part 4 of Def. 4.1 for $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively. We define M_{state} for $\mathcal{T}_1 \| \mathcal{T}_2$ as the probabilistic Turing machine that, given state $q = (q_1, q_2)$ of $\mathcal{T}_1 \| \mathcal{T}_2$ where $q_1 = q \lceil \mathcal{T}_1$ and $q_2 = q \lceil \mathcal{T}_2$ and action a, outputs the next state of $\mathcal{T}_1 \| \mathcal{T}_2$ as $q' = (q'_1, q'_2)$, where q'_1 is the next state of $\mathcal{T}_1$ and q'_2 is the next state of $\mathcal{T}_2$. The state q' is computed as follows:

   - If a is an action of $\mathcal{T}_1$ then q'_1 is the output of $M_{state1}(q_1, a)$, while $q'_1 = q_1$ otherwise.
   - If a is an action of $\mathcal{T}_2$ then q'_2 is the output of $M_{state2}(q_2, a)$, while $q'_2 = q_2$ otherwise.

   M_{state} always operates within time $\mathcal{O}(b_1 + b_2)$: this time is sufficient to determine whether a is an action of $\mathcal{T}_1$ and/or $\mathcal{T}_2$, to extract the needed parts of q to supply to M_{act1} and/or M_{act2}, and to run M_{state1} and/or M_{state2}.

Using standard Turing machine encodings, each of the needed Turing machines can be reprsented using $\mathcal{O}(b_1 + b_2)$ bits. □

For the rest of the paper, we fix some constant c_{comp} satisfying the conditions of Lemma 4.2.

4.3 Hiding

Lemma 4.3 *There exists a constant c such that the following holds. Suppose $\mathcal{T}$ is a b-time-bounded task-PIOA, where $b \in \mathsf{R}^{\geq 0}$, $b \geq 1$. Let $\mathcal{U}$ be a subset of the set of output tasks of $\mathcal{T}$, where $|\mathcal{U}| \leq c'$. Then $hide(\mathcal{T}, \mathcal{U})$ is a $c(c'+1)b$-time-bounded task-PIOA.*

Proof. All properties for $hide(\mathcal{T}, \mathcal{U})$ are straightforward to check, except for the following.

1. **Output actions:** To check whether a given action a is an output action of $hide(\mathcal{T}, \mathcal{U})$, we use the fact that a is an output action of $hide(\mathcal{T}, \mathcal{U})$ if and only if a is an output of $\mathcal{T}$ and is not in any task in $\mathcal{U}$. So, to determine whether a is an output of $hide(\mathcal{T}, \mathcal{U})$, we can use the procedure for checking whether a is an output of $\mathcal{T}$, followed by checking whether a is in each task in $\mathcal{U}$.

2. **Internal actions:** To check whether a given action a is an internal action of $hide(\mathcal{T},\mathcal{U})$, we use the fact that a is an internal action of $hide(\mathcal{T},\mathcal{U})$ if and only if a is an internal action of $\mathcal{T}$ or a is in some task in $\mathcal{U}$. So, to determine whether a is an internal action of $hide(\mathcal{T},\mathcal{U})$, we can use the procedure for checking whether a is an internal action of $\mathcal{T}$, followed by checking whether a is in each task in $\mathcal{U}$.

3. **Output tasks:** To check whether a given task T is an output task of $hide(\mathcal{T},\mathcal{U})$, we use the fact that T is an output task of $hide(\mathcal{T},\mathcal{U})$ if and only if T is an output task of $\mathcal{T}$ and $T \notin \mathcal{U}$. So, to determine whether T is an output task of $hide(\mathcal{T},\mathcal{U})$, we can use the procedure for checking whether T is an output task of $\mathcal{T}$, followed by comparing T with each task in $\mathcal{U}$. Each of these comparisons takes time proportional to b, which is a bound on the length of the tasks of $\mathcal{T}$.

4. **Internal tasks:** To check whether a given task T is an internal task of $hide(\mathcal{T},\mathcal{U})$, we use the fact that T is an internal task of $hide(\mathcal{T},\mathcal{U})$ if and only if T is an internal task of $\mathcal{T}$ or $T \in \mathcal{U}$. So, to determine whether T is an internal task of $hide(\mathcal{T},\mathcal{U})$, we can use the procedure for checking whether T is an internal task of $\mathcal{T}$, followed by comparing T with each task in $\mathcal{U}$. Again, each of these comparisons takes time proportional to b which is a bound on the length of the tasks of $\mathcal{T}$.

In all cases, the total time is proportional to $(c' + 1)b$. Using standard Turing machine encodings, each of the needed Turing machines can be represented using $\mathcal{O}(b_1 + b_2)$ bits. □

For the rest of this paper, we fix some constant c_{hide} satisfying the conditions of Lemma 4.3.

4.4 Time-Bounded Task Scheduler

Definition 4.4 *Let ρ be a task scheduler for closed task-PIOA $\mathcal{T}$, and let $b \in \mathsf{N}$. Then we say that ρ is* b-time-bounded *if $|\rho| \leq b$, that is, if the number of tasks in the task scheduler ρ is at most b.*

4.5 Implementation

In Section 3.7, we defined an implementation relation $\leq_0$ for task-PIOAs. Informally speaking, for task-PIOAs $\mathcal{T}_1$ and $\mathcal{T}_2$, $\mathcal{T}_1 \leq_0 \mathcal{T}_2$ means that $\mathcal{T}_1$ "looks the same" as $\mathcal{T}_2$, to any environment $\mathcal{E}$. Here, "looking the same" means that any trace distribution of $\mathcal{T}_1 \| \mathcal{E}$ is also a trace distribution of $\mathcal{T}_2 \| \mathcal{E}$.

Now we define another implementation relation, $\leq_{\epsilon,b,b_1,b_2}$, for task-PIOAs that allows some discrepancies in the trace distributions and also takes time bounds into account. Informally speaking, $\mathcal{T}_1 \leq_{\epsilon,b,b_1,b_2} \mathcal{T}_2$ means that $\mathcal{T}_1$ "looks almost the same" as task-PIOA $\mathcal{T}_2$ to any b-time-bounded environment $\mathcal{E}$. The subscripts b_1 and b_2 in the relation $\leq_{\epsilon,b,b_1,b_2}$ represent time bounds on task schedulers. Namely, in the definition of $\leq_{\epsilon,b,b_1,b_2}$, we assume that scheduling in $\mathcal{T}_1 \| \mathcal{E}$ is controlled by a b_1-time-bounded task scheduler, and require that scheduling in $\mathcal{T}_2 \| \mathcal{E}$ be controlled by a b_2-bounded task scheduler. The fact that these task-PIOAs look "almost the same" is observed through the special *accept* output of $\mathcal{E}$:

Definition 4.5 *If $\mathcal{T}$ is a closed task-PIOA and ρ is a task scheduler for $\mathcal{T}$, then we define*

$$Paccept(\mathcal{T}, \rho) = \Pr[\beta \leftarrow tdist(\mathcal{T}, \rho) : \beta \text{ contains } accept],$$

that is, the probability that a trace chosen randomly from the trace distribution generated by ρ contains the accept output action.

Definition 4.6 *Suppose $\mathcal{T}_1$ and $\mathcal{T}_2$ are comparable task-PIOAs, $\epsilon, b \in \mathsf{R}^{\geq 0}$, and $b_1, b_2 \in \mathsf{N}$. Then we say that $\mathcal{T}_1 \leq_{\epsilon,b,b_1,b_2} \mathcal{T}_2$ provided that, for every b-time-bounded environment $\mathcal{E}$ for both $\mathcal{T}_1$ and $\mathcal{T}_2$, and for every b_1-time-bounded task scheduler ρ_1 for $\mathcal{T}_1 \| \mathcal{E}$, there is a b_2-time-bounded task scheduler ρ_2 for $\mathcal{T}_2 \| \mathcal{E}$ such that*

$$|Paccept(\mathcal{T}_1 \| \mathcal{E}, \rho_1) - Paccept(\mathcal{T}_2 \| \mathcal{E}, \rho_2)| \leq \epsilon.$$

A useful property of the $\leq_{\epsilon,b,b_1,b_2}$ relation is that it is transitive:

Lemma 4.7 *Suppose $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ are three comparable task-PIOAs such that $\mathcal{T}_1 \leq_{\epsilon_{12},b,b_1,b_2} \mathcal{T}_2$ and $\mathcal{T}_2 \leq_{\epsilon_{23},b,b_2,b_3} \mathcal{T}_3$, where $\epsilon, b \in \mathsf{R}^{\geq 0}$ and $b_1, b_2, b_3 \in \mathsf{N}$.*
Then $\mathcal{T}_1 \leq_{\epsilon_{12}+\epsilon_{23},b,b_1,b_3} \mathcal{T}_3$.

Proof. Fix $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{T}_3$ and all the constants as in the hypotheses. Consider any b-time-bounded environment $\mathcal{E}$ for $\mathcal{T}_1$ and $\mathcal{T}_3$. We must show that, for every b_1-time-bounded task scheduler ρ_1 for $\mathcal{T}_1$, there is a b_3-time-bounded task scheduler ρ_3 for $\mathcal{T}_3$ such that

$$|Paccept(\mathcal{T}_1\|\mathcal{E},\rho_1) - Paccept(\mathcal{T}_3\|\mathcal{E},\rho_3)| \leq \epsilon_{12} + \epsilon_{23}.$$

Fix ρ_1 to be any b_1-time-bounded task scheduler for $\mathcal{T}_1$. We consider two cases.

First, suppose that $\mathcal{E}$ is also an environment for $\mathcal{T}_2$. Then, since $\mathcal{T}_1 \leq_{\epsilon_{12},b,b_1,b_2} \mathcal{T}_2$, we know that there is a b_2-time-bounded task scheduler ρ_2 for $\mathcal{T}_2\|\mathcal{E}$ such that

$$|Paccept(\mathcal{T}_1\|\mathcal{E},\rho_1) - Paccept(\mathcal{T}_2\|\mathcal{E},\rho_2)| \leq \epsilon_{12}.$$

Then since $\mathcal{T}_2 \leq_{\epsilon_{23},b,b_2,b_3} \mathcal{T}_3$, we may conclude that there is a b_3-time-bounded task scheduler ρ_3 for $\mathcal{T}_3\|\mathcal{E}$ such that

$$|Paccept(\mathcal{T}_2\|\mathcal{E},\rho_2) - Paccept(\mathcal{T}_3\|\mathcal{E},\rho_3)| \leq \epsilon_{23}.$$

Combining these two properties, we obtain that:

$$\begin{aligned}
&|Paccept(\mathcal{T}_1\|\mathcal{E},\rho_1) - Paccept(\mathcal{T}_3\|\mathcal{E},\rho_3)| \\
&\qquad \leq |Paccept(\mathcal{T}_1\|\mathcal{E},\rho_1) - Paccept(\mathcal{T}_2\|\mathcal{E},\rho_2)]| \\
&\qquad\quad +|Paccept(\mathcal{T}_2\|\mathcal{E},\rho_2) - Paccept(\mathcal{T}_3\|\mathcal{E},\rho_3)]| \\
&\qquad \leq \epsilon_{12} + \epsilon_{23},
\end{aligned}$$

as needed.

Second, consider the case where $\mathcal{E}$ is not an environment for $\mathcal{T}_2$. This may be for two reasons: first, because $\mathcal{E}$ has internal actions that are actions of $\mathcal{T}_2$; second, because $\mathcal{E}$ has output actions that are internal or output actions of $\mathcal{T}_2$. In both cases, by an appropriate renaming of the internal and output actions of $\mathcal{E}$, we can transform this environment into an environment $\mathcal{E}'$ for $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ such that, for every schedulers ρ_1 for $\mathcal{T}_1$ and ρ_3 for $\mathcal{T}_3$, we have:

$$Paccept(\mathcal{T}_1\|\mathcal{E},\rho_1) = Paccept(\mathcal{T}_1\|\mathcal{E}',\rho_1)$$

and

$$Paccept(\mathcal{T}_3\|\mathcal{E},\rho_3) = Paccept(\mathcal{T}_3\|\mathcal{E}',\rho_3).$$

Now, we apply case 1 to $\mathcal{E}'$, obtaining a b_3-time-bounded task scheduler ρ_3 for $\mathcal{T}_3$ such that

$$|Paccept(\mathcal{T}_1\|\mathcal{E}',\rho_1) - Paccept(\mathcal{T}_3\|\mathcal{E}',\rho_3)| \leq \epsilon_{12} + \epsilon_{23}.$$

Therefore,

$$\begin{aligned}
&|Paccept(\mathcal{T}_1\|\mathcal{E},\rho_1) - Paccept(\mathcal{T}_3\|\mathcal{E},\rho_3)| \\
&\qquad = |Paccept(\mathcal{T}_1\|\mathcal{E}',\rho_1) - Paccept(\mathcal{T}_3\|\mathcal{E}',\rho_3)| \\
&\qquad \leq \epsilon_{12} + \epsilon_{23},
\end{aligned}$$

as needed. □

Another useful property of the $\leq_{\epsilon,b,b_1,b_2}$ relation is that, under certain conditions, it is preserved under composition:

Lemma 4.8 *Suppose $\epsilon, b, b_3 \in \mathsf{R}^{\geq 0}$, and $b_1, b_2 \in \mathsf{N}$. Suppose that $\mathcal{T}_1$, $\mathcal{T}_2$ are comparable task-PIOAs such that $\mathcal{T}_1 \leq_{\epsilon, c_{comp}(b+b_3), b_1, b_2} \mathcal{T}_2$. Suppose that $\mathcal{T}_3$ is a b_3-time-bounded task-PIOA that is compatible with both $\mathcal{T}_1$ and $\mathcal{T}_2$.*
Then $\mathcal{T}_1 \| \mathcal{T}_3 \leq_{\epsilon, b, b_1, b_2} \mathcal{T}_2 \| \mathcal{T}_3$.

Proof. Fix $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ and all the constants as in the hypotheses. Consider any b-time-bounded environment $\mathcal{E}$ for $\mathcal{T}_1 \| \mathcal{T}_3$ and $\mathcal{T}_2 \| \mathcal{T}_3$. We must show that, for every b_1-time-bounded task scheduler ρ_1 for $\mathcal{T}_1 \| \mathcal{T}_3$, there is a b_2-time-bounded task scheduler ρ_2 for $\mathcal{T}_2 \| \mathcal{T}_3$ such that

$$|Paccept(\mathcal{T}_1 \| \mathcal{T}_3 \| \mathcal{E}, \rho_1) - Paccept(\mathcal{T}_2 \| \mathcal{T}_3 \| \mathcal{E}, \rho_2)| \leq \epsilon.$$

To show this, fix ρ_1 to be any b_1-time-bounded task scheduler for $\mathcal{T}_1 \| \mathcal{T}_3$. The composition $\mathcal{T}_3 \| \mathcal{E}$ is an environment for $\mathcal{T}_1$ and $\mathcal{T}_2$. Moreover, Lemma 4.2 implies that $\mathcal{T}_3 \| \mathcal{E}$ is $c_{comp}(b + b_3)$-time-bounded.

Since $\mathcal{T}_1 \leq_{\epsilon, c_{comp}(b+b_3), b_1, b_2} \mathcal{T}_2$, $\mathcal{T}_3 \| \mathcal{E}$ is a $c_{comp}(b + b_3)$-time-bounded environment for $\mathcal{T}_1$ and $\mathcal{T}_2$, and ρ_1 is a b_1-time-bounded task scheduler for $\mathcal{T}_1 \| \mathcal{E}$, we know that there is a b_2-time-bounded task scheduler ρ_2 for $\mathcal{T}_2 \| \mathcal{E}$ such that

$$|Paccept(\mathcal{T}_1 \| \mathcal{T}_3 \| \mathcal{E}, \rho_1) - Paccept(\mathcal{T}_2 \| \mathcal{T}_3 \| \mathcal{E}, \rho_2)| \leq \epsilon.$$

This is as needed. □

One last interesting property of our $\leq_{\epsilon, b, b_1, b_2}$ relation is that it is preserved when hiding output actions of the related task-PIOAs:

Lemma 4.9 *Suppose $\epsilon, b \in \mathsf{R}^{\geq 0}$, and $b_1, b_2 \in \mathsf{N}$. Suppose that $\mathcal{T}_1$, $\mathcal{T}_2$ are comparable task-PIOAs such that $\mathcal{T}_1 \leq_{\epsilon, b, b_1, b_2} \mathcal{T}_2$. Suppose also that $\mathcal{U}$ is a set of output tasks of both $\mathcal{T}_1$ and $\mathcal{T}_2$.*
Then $hide(\mathcal{T}_1, \mathcal{U}) \leq_{\epsilon, b, b_1, b_2} hide(\mathcal{T}_2, \mathcal{U})$.

Proof. This follows from the fact that every b-bounded environment for $hide(\mathcal{T}_1, \mathcal{U})$ and $hide(\mathcal{T}_2, \mathcal{U})$ is also a b-bounded environment for $\mathcal{T}_1$ and $\mathcal{T}_2$. □

4.6 Simulation Relations

The simulation relation we defined in Section 3.8 can be applied to time-bounded task-PIOAs. We obtain the following additional soundness theorem:

Theorem 4.10 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two comparable task-PIOAs, $b \in \mathsf{R}^{\geq 0}$, and $c, b_1 \in \mathsf{N}$. Suppose that, for every b-bounded environment $\mathcal{E}$ for $\mathcal{T}_1$ and $\mathcal{T}_2$, there exists a simulation relation from $\mathcal{T}_1 \| \mathcal{E}$ to $\mathcal{T}_2 \| \mathcal{E}$ using corrtasks, and $|corrtasks(\rho, T)| \leq c$ for every ρ and T.*
Then $\mathcal{T}_1 \leq_{0, b, b_1, cb_1} \mathcal{T}_2$.

Proof. By Theorem 3.30 and the definition of our new implementation relationship. □

4.7 Task-PIOA Families

Here we define families of task-PIOAs, and define what it means for a family of task-PIOAs to be time-bounded by a function of the index of the family.

4.7.1 Basic Definitions

A *task-PIOA family*, $\overline{\mathcal{T}}$, is an indexed set, $\{\mathcal{T}_k\}_{k \in \mathsf{N}}$, of task-PIOAs. A task-PIOA family $\overline{\mathcal{T}} = \{\mathcal{T}_k\}_{k \in \mathsf{N}}$ is said to be *closed* provided that, for every k, $\mathcal{T}_k$ is closed.

Two task-PIOA families $\overline{\mathcal{T}}_1 = \{(\mathcal{T}_1)_k\}_{k \in \mathsf{N}}$ and $\overline{\mathcal{T}}_2 = \{(\mathcal{T}_2)_k\}_{k \in \mathsf{N}}$ are said to be *comparable* provided that, for every k, $(\mathcal{T}_1)_k$ and $(\mathcal{T}_2)_k$ are comparable.

Two task-PIOA families $\overline{\mathcal{T}}_1 = \{(\mathcal{T}_1)_k\}_{k \in \mathsf{N}}$ and $\overline{\mathcal{T}}_2 = \{(\mathcal{T}_2)_k\}_{k \in \mathsf{N}}$ are said to be *compatible* provided that, for every k, $(\mathcal{T}_1)_k$ and $(\mathcal{T}_2)_k$ are compatible. Two compatible task-PIOA families $\overline{\mathcal{T}}_1 = \{(\mathcal{T}_1)_k\}_{k \in \mathsf{N}}$ and $\overline{\mathcal{T}}_2 = \{(\mathcal{T}_2)_k\}_{k \in \mathsf{N}}$ can be composed to yield $\overline{\mathcal{T}} = \{(\mathcal{T})_k\}_{k \in \mathsf{N}} = \overline{\mathcal{T}}_1 \| \overline{\mathcal{T}}_2$ by defining $(\mathcal{T})_k = (\mathcal{T}_1)_k \| (\mathcal{T}_2)_k$ for every k.

Definition 4.11 *A* task-set family *for a task-PIOA family* $\overline{\mathcal{T}} = \{\mathcal{T}_k\}_{k\in\mathsf{N}}$ *is an indexed set,* $\overline{\mathcal{U}} = \{\mathcal{U}_k\}_{k\in\mathsf{N}}$*, where each* $\mathcal{U}_k$ *is a set of tasks of* $\mathcal{T}_k$*. We say that* $\overline{\mathcal{U}}$ *is an* output-task-set family *if each* $\mathcal{U}_k$ *is a set of output tasks of* $\mathcal{T}_k$*.*

If $\overline{\mathcal{T}}$ *is a task-PIOA family and* $\mathcal{U}$ *is an output-task-set family for* $\overline{\mathcal{T}}$*, then we define* $hide(\overline{\mathcal{T}},\mathcal{U})$ *to be the family* $(hide(T_k,\mathcal{U}_k))_{k\in\mathsf{N}}$*.*

A *task-scheduler family* $\overline{\rho}$ for a closed task-PIOA family $\overline{\mathcal{T}} = \{\mathcal{T}_k\}_{k\in\mathsf{N}}$ is an indexed set, $\{\rho_k\}_{k\in\mathsf{N}}$ of task schedulers, where ρ_k is a task scheduler for $\mathcal{T}_k$.

4.7.2 Time-Bounded Task-PIOA Families

Definition 4.12 *The task-PIOA family* $\overline{\mathcal{T}} = \{\mathcal{T}_k\}_{k\in\mathsf{N}}$ *is said to be* b-time-bounded *(or* non-uniformly b-time bounded*), where* $b : \mathsf{N} \to \mathsf{R}^{\geq 0}$*, provided that* $\mathcal{T}_k$ *is* $b(k)$*-time bounded for every* k*.*

This definition allows different Turing machines to be used for each k. In some situations, we will add a uniformity condition requiring the same Turing machines to be used for all task-PIOAs of the family; these machines receive k as an auxiliary input.

Definition 4.13 *The task-PIOA family* $\overline{\mathcal{T}} = \{\mathcal{T}_k\}_{k\in\mathsf{N}}$ *is said to be* uniformly b-time-bounded*, where* $b : \mathsf{N} \to \mathsf{R}^{\geq 0}$*, provided that:*

1. $\mathcal{T}_k$ *is* $b(k)$*-bounded for every* k*.*
2. *There is a deterministic Turing machine that, given* k *and a candidate state* q*, decides whether* q *is a state of* $\mathcal{T}_k$*, and always runs in time at most* $b(k)$*. Similarly for a candidate start state, input action, output action, internal action, transition, output task or internal task. Also, there is a deterministic Turing machine that, given* k *and two candidate actions* a_1 *and* a_2*, decides whether* $(a_1, a_2) \in R_{\mathcal{T}_k}$*, and always runs in time at most* $b(k)$*. Also, there is a deterministic Turing machine that, given* k*, an action* a *of* $\mathcal{T}_k$ *and a task* T*, decides whether* $a \in T$*; again this machine runs in time at most* $b(k)$*.*
3. *There is a deterministic Turing machine* M_{act} *that, given* k*, state* q *of* $\mathcal{T}_k$ *and an output or internal task* T *of* $\mathcal{T}_k$*, produces the unique action* a *in task* T *that is enabled in* q *if one exists, and otherwise produces a special "no-action" indicator. Moreover,* M_{act} *always runs in time at most* $b(k)$*.*
4. *There is a probabilistic Turing machine* M_{state} *that, given* k*, state* q *of* $\mathcal{T}_k$*, and the representation of an action* a *of* $\mathcal{T}_k$ *that is enabled in* q*, produces the next state resulting from the unique transition of* $\mathcal{T}_k$ *of the form* (q, a, μ)*. Moreover,* M_{state} *always runs in time at most* $b(k)$*.*

Lemma 4.14 *Suppose* $\overline{\mathcal{T}}_1$ *and* $\overline{\mathcal{T}}_2$ *are two compatible task-PIOA families,* $\overline{\mathcal{T}}_1$ *is* b_1*-time-bounded, and* $\overline{\mathcal{T}}_2$ *is* b_2*-time-bounded, where* $b_1, b_2 : \mathsf{N} \to \mathsf{R}^{\geq 0}$*. Then* $\overline{\mathcal{T}}_1 \| \overline{\mathcal{T}}_2$ *is a* $c_{comp}(b_1+b_2)$*-time-bounded task-PIOA family.*

Proof. By Lemma 4.2 and the definition of a time-bounded task-PIOA family. □

Lemma 4.15 *Suppose* $\overline{\mathcal{T}}$ *is a* b*-time-bounded task-PIOA family, where* $b : \mathsf{N} \to \mathsf{R}^{\geq 0}$*. Suppose that* $\overline{\mathcal{U}} = \{\mathcal{U}_k\}_{k\in\mathsf{N}}$ *is a task-set family for* $\overline{\mathcal{T}}$*, where each* $\mathcal{U}_k$ *is a set of output tasks for* $\mathcal{T}_k$ *with* $|\mathcal{U}_k| \leq c$*. Then* $hide(\overline{\mathcal{T}},\overline{\mathcal{U}})$ *is a* $c_{hide}(c+1)b$*-time-bounded task-PIOA family.*

Proof. By Lemma 4.3. □

Definition 4.16 *Let* $\overline{\rho} = \{\rho_k\}_{k\in\mathsf{N}}$ *be a task-scheduler family for a closed task-PIOA family* $\overline{\mathcal{T}} = \{\mathcal{T}_k\}_{k\in\mathsf{N}}$*. Then* $\overline{\rho}$ *is said to be* b-time-bounded*, where* $b : \mathsf{N} \to \mathsf{R}^{\geq 0}$ *provided that* ρ_k *is* $b(k)$*-time bounded for every* k*.*

Now we extend the time-bounded implementation notion to task-PIOA families:

Definition 4.17 *Suppose $\overline{\mathcal{T}}_1 = \{(\mathcal{T}_1)_k\}_{k\in\mathsf{N}}$ and $\overline{\mathcal{T}}_2 = \{(\mathcal{T}_2)_k\}_{k\in\mathsf{N}}$ are comparable task-PIOA families and ϵ, b, b_1 and b_2 are functions, where $\epsilon, b : \mathsf{N} \to \mathsf{R}^{\geq 0}$, and $b_1, b_2 : \mathsf{N} \to \mathsf{N}$. Then we say that $\overline{\mathcal{T}}_1 \leq_{\epsilon,b,b_1,b_2} \overline{\mathcal{T}}_2$ provided that $(\mathcal{T}_1)_k \leq_{\epsilon(k),b(k),b_1(k),b_2(k)} (\mathcal{T}_2)_k$ for every k.*

Our previous transitivity result for individual automata carries over to families:

Lemma 4.18 *Suppose $\overline{\mathcal{T}}_1$, $\overline{\mathcal{T}}_2$ and $\overline{\mathcal{T}}_3$ are three comparable task-PIOA families such that $\overline{\mathcal{T}}_1 \leq_{\epsilon_{12},b,b_1,b_2} \overline{\mathcal{T}}_2$ and $\overline{\mathcal{T}}_2 \leq_{\epsilon_{23},b,b_2,b_3} \overline{\mathcal{T}}_3$, where $\epsilon, b : \mathsf{N} \to \mathsf{R}^{\geq 0}$ and $b_1, b_2 : \mathsf{N} \to \mathsf{N}$.*
Then $\overline{\mathcal{T}}_1 \leq_{\epsilon_{12}+\epsilon_{23},b,b_1,b_3} \overline{\mathcal{T}}_3$.

Proof. Suppose $\overline{\mathcal{T}}_1 = \{(\mathcal{T}_1)_k\}_{k\in\mathsf{N}}$, $\overline{\mathcal{T}}_2 = \{(\mathcal{T}_2)_k\}_{k\in\mathsf{N}}$ and $\overline{\mathcal{T}}_3 = \{(\mathcal{T}_3)_k\}_{k\in\mathsf{N}}$ are three comparable task-PIOA families satisfying the hypotheses. Then Definition 4.17 implies that, for every k, $(\mathcal{T}_1)_k \leq_{\epsilon_{12}(k),b(k),b_1(k),b_2(k)} (\mathcal{T}_2)_k$ and $(\mathcal{T}_2)_k \leq_{\epsilon_{23}(k),b(k),b_2(k),b_3(k)} (\mathcal{T}_3)_k$. Lemma 4.7 then implies that, for every k, $(\mathcal{T}_1)_k \leq_{\epsilon_{12}(k)+\epsilon_{23}(k),b(k),b_1(k),b_3(k)} (\mathcal{T}_3)_k$. Applying Definition 4.17 once again, we obtain that $\overline{\mathcal{T}}_1 \leq_{\epsilon_{12}+\epsilon_{23},b,b_1,b_3} \overline{\mathcal{T}}_3$, as needed. □

Our previous composition result for individual automata also carries over to families:

Lemma 4.19 *Suppose $\epsilon, b, b_3 : \mathsf{N} \to \mathsf{R}^{\geq 0}$, and $b_1, b_2 : \mathsf{N} \to \mathsf{N}$. Suppose $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ are comparable families of task-PIOAs such that $\overline{\mathcal{T}}_1 \leq_{\epsilon,c_{comp}(b+b_3),b_1,b_2} \overline{\mathcal{T}}_2$. Suppose that $\overline{\mathcal{T}}_3$ is a b_3-time-bounded task-PIOA family that is compatible with both $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$.*
Then $\overline{\mathcal{T}}_1 \| \overline{\mathcal{T}}_3 \leq_{\epsilon,b,b_1,b_2} \overline{\mathcal{T}}_2 \| \overline{\mathcal{T}}_3$.

Proof. Fix $\overline{\mathcal{T}}_1 = \{(\mathcal{T}_1)_k\}_{k\in\mathsf{N}}$, $\overline{\mathcal{T}}_2 = \{(\mathcal{T}_2)_k\}_{k\in\mathsf{N}}$, $\overline{\mathcal{T}}_3 = \{(\mathcal{T}_3)_k\}_{k\in\mathsf{N}}$ and all the functions as in the hypotheses. By Definition 4.17, for every k, $(\mathcal{T}_1)_k \leq_{\epsilon(k),c_{comp}(b+b_3)(k),b_1(k),b_2(k)} (\mathcal{T}_2)_k$. Lemma 4.8 then implies that, for every k, $(\mathcal{T}_1)_k \| (\mathcal{T}_3)_k \leq_{\epsilon(k),b(k),b_1(k),b_2(k)} (\mathcal{T}_2)_k \| (\mathcal{T}_3)_k$. Applying Definition 4.17 once again, we obtain that $\overline{\mathcal{T}}_1 \| \overline{\mathcal{T}}_3 \leq_{\epsilon,b,b_1,b_2} \overline{\mathcal{T}}_2 \| \overline{\mathcal{T}}_3$, as needed. □

Hiding output actions of task-PIOA families also preserves the new relation:

Lemma 4.20 *Suppose $\epsilon, b : \mathsf{N} \to \mathsf{R}^{\geq 0}$, and $b_1, b_2 : \mathsf{N} \to \mathsf{N}$. Suppose that $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ are comparable task-PIOA families such that $\overline{\mathcal{T}}_1 \leq_{\epsilon,b,b_1,b_2} \overline{\mathcal{T}}_2$. Suppose that $\mathcal{U}$ is an output-task-set family for both $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$.*
Then $hide(\overline{\mathcal{T}}_1, \mathcal{U}) \leq_{\epsilon,b,b_1,b_2} hide(\overline{\mathcal{T}}_2, \mathcal{U})$.

Proof. By Lemma 4.9. □

Finally, we obtain a soundness result for simulation relations:

Theorem 4.21 *Let $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ be comparable task-PIOA families, $c \in \mathsf{N}$, $b : \mathsf{N} \to \mathsf{R}^{\geq 0}$, and $b_1 : \mathsf{N} \to \mathsf{N}$. Suppose that, for every k, and for every $b(k)$-bounded environment $\mathcal{E}$ for $(\mathcal{T}_1)_k$ and $(\mathcal{T}_2)_k$, there exists a simulation relation from $(\mathcal{T}_1)_k \| \mathcal{E}$ to $(\mathcal{T}_2)_k \| \mathcal{E}$ using corrtasks, and $|corrtasks(\rho, T)| \leq c$ for every ρ and T.*
Then $\overline{\mathcal{T}}_1 \leq_{0,b,b_1,cb_1} \overline{\mathcal{T}}_2$.

Proof. By Theorem 4.10. □

4.7.3 Polynomial-Time Task-PIOA Families

Definition 4.22 *The task-PIOA family $\overline{\mathcal{T}}$ is said to be* polynomial-time-bounded *(or* non-uniformly polynomial-time-bounded*) provided that there exists a polynomial p such that $\overline{\mathcal{T}}$ is p-time-bounded.*

$\overline{\mathcal{T}}$ is said to be uniformly polynomial-time-bounded *provided that there exists a polynomial p such that $\overline{\mathcal{T}}$ is uniformly p-time-bounded.*

Lemma 4.23 *Suppose $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ are two compatible polynomial-time-bounded task-PIOA families. Then $\overline{\mathcal{T}}_1 \| \overline{\mathcal{T}}_2$ is a polynomial-time-bounded task-PIOA family.*

Proof. Suppose p_1 and p_2 are polynomials such that $\overline{\mathcal{T}}_1$ is p_1-time-bounded and $\overline{\mathcal{T}}_2$ is p_2-time-bounded. Then by Lemma 4.2, Then $\overline{\mathcal{T}}_1 \| \overline{\mathcal{T}}_2$ is $c_{comp}(p_1 + p_2)$-time-bounded, which implies that it is polynomial-time-bounded. □

Lemma 4.24 *Suppose $\overline{\mathcal{T}}$ is a polynomial-time-bounded task-PIOA family. Suppose that $\overline{\mathcal{U}} = \{\mathcal{U}_k\}_{k \in \mathsf{N}}$ is a task-set family for $\overline{\mathcal{T}}$, where each $\mathcal{U}_k$ is a set of output tasks for $\mathcal{T}_k$ with $|\mathcal{U}_k| \leq c$. Then hide$(\overline{\mathcal{T}}, \overline{\mathcal{U}})$ is a polynomial-time-bounded task-PIOA family.*

Proof. By Lemma 4.15. □

Definition 4.25 *Let $\overline{\rho} = \{\rho_k\}_{k \in \mathsf{N}}$ be a task-scheduler family for a closed task-PIOA family $\overline{\mathcal{T}} = \{\mathcal{T}_k\}_{k \in \mathsf{N}}$. Then $\overline{\rho}$ is said to be* polynomial time-bounded *provided that there exists a polynomial p such that $\overline{\rho}$ is p-time-bounded.*

In the context of cryptography, we will want to say that, for every polynomial-time-bounded environment, the probability of distinguishing two systems is "negligible". The notion of negligible probability is expressed by saying that the that the probability must be less than a negligible function ϵ:

Definition 4.26 *A function ϵ is said to be* negligible *iff, for every constant $c \in \mathbf{R}^+$, there exists k_0 such that, $\forall k \geq k_0$, $\epsilon(k) < \frac{1}{k^c}$.*

Definition 4.27 *Suppose $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ are comparable task-PIOA families. We say that $\overline{\mathcal{T}}_1 \leq_{neg,pt} \overline{\mathcal{T}}_2$ iff, for every polynomial p and polynomial p_1, there is a polynomial p_2 and a negligible function ϵ such that $\overline{\mathcal{T}}_1 \leq_{\epsilon,p,p_1,p_2} \overline{\mathcal{T}}_2$.*

Lemma 4.28 *Suppose $\overline{\mathcal{T}}_1$, $\overline{\mathcal{T}}_2$ and $\overline{\mathcal{T}}_3$ are three comparable task-PIOA families such that $\overline{\mathcal{T}}_1 \leq_{neg,pt} \overline{\mathcal{T}}_2$ and $\overline{\mathcal{T}}_2 \leq_{neg,pt} \overline{\mathcal{T}}_3$.*
Then $\overline{\mathcal{T}}_1 \leq_{neg,pt} \overline{\mathcal{T}}_3$.

Proof. Suppose $\overline{\mathcal{T}}_1 = \{(\mathcal{T}_1)_k\}_{k \in \mathsf{N}}$, $\overline{\mathcal{T}}_2 = \{(\mathcal{T}_2)_k\}_{k \in \mathsf{N}}$ and $\overline{\mathcal{T}}_3 = \{(\mathcal{T}_3)_k\}_{k \in \mathsf{N}}$ are three comparable task-PIOA families satisfying the hypotheses. To show that $\overline{\mathcal{T}}_1 \leq_{neg,pt} \overline{\mathcal{T}}_3$, we fix polynomials p and p_1; we must obtain a polynomial p_3 and a negligible function ϵ_{13} such that $\overline{\mathcal{T}}_1 \leq_{\epsilon_{13},p,p_1,p_3} \overline{\mathcal{T}}_3$.

Since $\overline{\mathcal{T}}_1 \leq_{neg,pt} \overline{\mathcal{T}}_2$, we know that there exist polynomial p_2 and negligible function ϵ_{12} such that $\overline{\mathcal{T}}_1 \leq_{\epsilon_{12},p,p_1,p_2} \overline{\mathcal{T}}_3$. Then since $\overline{\mathcal{T}}_2 \leq_{neg,pt} \overline{\mathcal{T}}_3$, we may conclude that there exist polynomial p_3 and negligible function ϵ_{23} such that $\overline{\mathcal{T}}_1 \leq_{\epsilon_{23},p,p_2,p_3} \overline{\mathcal{T}}_3$. Let $\epsilon_{13} = \epsilon_{12} + \epsilon_{23}$. Then Lemma 4.18 implies that $\overline{\mathcal{T}}_1 \leq_{\epsilon_{13},p,p_1,p_3} \overline{\mathcal{T}}_3$, as needed. □

The $\leq_{neg,pt}$ relation is also preserved under composition with polynomial-time bounded task-PIOA families.

Lemma 4.29 *Suppose $\overline{\mathcal{T}}_1$, $\overline{\mathcal{T}}_2$ are comparable families of task-PIOAs such that $\overline{\mathcal{T}}_1 \leq_{neg,pt} \overline{\mathcal{T}}_2$, and suppose $\overline{\mathcal{T}}_3$ is a polynomial time-bounded task-PIOA family, compatible with both $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$.*
Then $\overline{\mathcal{T}}_1 \| \overline{\mathcal{T}}_3 \leq_{neg,pt} \overline{\mathcal{T}}_2 \| \overline{\mathcal{T}}_3$.

Proof. Suppose $\overline{\mathcal{T}}_1 = \{(\mathcal{T}_1)_k\}_{k\in\mathsf{N}}$, $\overline{\mathcal{T}}_2 = \{(\mathcal{T}_2)_k\}_{k\in\mathsf{N}}$, and $\overline{\mathcal{T}}_3 = \{(\mathcal{T}_3)_k\}_{k\in\mathsf{N}}$ are as in the hypotheses. Fix polynomial q such that $\overline{\mathcal{T}}_3$ is q-time-bounded. To show that $\overline{\mathcal{T}}_1 \| \overline{\mathcal{T}}_3 \leq_{neg,pt} \overline{\mathcal{T}}_2 \| \overline{\mathcal{T}}_3$, we fix polynomials p and p_1; we must obtain a polynomial p_2 and a negligible function ϵ such that $\overline{\mathcal{T}}_1 \| \overline{\mathcal{T}}_3 \leq_{\epsilon,p,p_1,p_2} \overline{\mathcal{T}}_2 \| \overline{\mathcal{T}}_3$.

Define p' to be the polynomial $c_{comp}(p+q)$. Since $\overline{\mathcal{T}}_1 \leq_{neg,pt} \overline{\mathcal{T}}_2$, there exist a polynomial p_2 and a negligible function ϵ such that $\overline{\mathcal{T}}_1 \leq_{\epsilon,p',p_1,p_2} \overline{\mathcal{T}}_2$. Lemma 4.19 then implies that $\overline{\mathcal{T}}_1 \| \overline{\mathcal{T}}_3 \leq_{\epsilon,p,p_1,p_2} \overline{\mathcal{T}}_2 \| \overline{\mathcal{T}}_3$, as needed. □

Hiding output actions of the task-PIOAs that we compare also preserves the $\leq_{neg,pt}$ relation.

Lemma 4.30 *Suppose that $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ are comparable task-PIOA families such that $\overline{\mathcal{T}}_1 \leq_{neg,pt} \overline{\mathcal{T}}_2$. Suppose that $\overline{\mathcal{U}}$ is an output-task-set family for both $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$.*
Then $hide(\overline{\mathcal{T}}_1, \overline{\mathcal{U}}) \leq_{neg,pt} hide(\overline{\mathcal{T}}_2, \mathcal{U})$.

Proof. By Lemma 4.20. □

And we have another soundness result for simulation relations:

Theorem 4.31 *Let $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ be comparable task-PIOA families, $c \in \mathsf{N}$.*
Suppose that for every polynomial p, for every k, and for every $p(k)$-bounded environment $\mathcal{E}$ for $(\mathcal{T}_1)_k$ and $(\mathcal{T}_2)_k$, there exists a simulation relation from $(\mathcal{T}_1)_k \| \mathcal{E}$ to $(\mathcal{T}_2)_k \| \mathcal{E}$ using corrtasks, for which $|corrtasks(\rho, T)| \leq c$ for every ρ and T.
Then $\overline{\mathcal{T}}_1 \leq_{neg,pt} \overline{\mathcal{T}}_2$.

Proof. By Theorem 4.21. □

5 Random Source Automata

We now present a first example of task-PIOA. We will sometimes find it convenient to separate out random choices into separate "random source" components. One type of random source is one that simply chooses and outputs a single value, obtained from a designated probability distribution. We define this type of source by a task-PIOA $Src(D, \mu)$, parameterized by a probability distribution (D, μ). When μ is the uniform distribution over D, we write simply $Src(D)$.

The code for task-PIOA $Src(D, \mu)$ appears in Figure 1.

We extend this definition to indexed families of data types and distributions, $D = \{D_k\}_{k\in\mathsf{N}}$ and $\mu = \{\mu_k\}_{k\in\mathsf{N}}$, to yield an indexed family of random source automata, $Src(D, \mu) = \{Src(D_k, \mu_k)\}_{k\in\mathsf{N}}$. As before, when every μ_k is the uniform distribution, we write simply $Src(D) = \{Src(D_k)\}_{k\in\mathsf{N}}$.

6 Hard-Core Predicates

In this section, we define a cryptographic primitive—a hard-core predicate for a trap-door permutation—that we use in several of our system descriptions. We define this in terms of task-PIOAs, and relate the new definition to the standard cryptographic definition. Using our new task-PIOA formulation, we show some consequences of the definition, in particular, we show how a hard-core predicate retains its properties if it is used twice, and if it is combined with another value using an $\oplus$ operation.

Throughout this section, we fix $\overline{D} = \{D_k\}_{k\in\mathsf{N}}$ to be a family of finite domains, and $\overline{Tdp} = \{Tdp_k\}_{k\in\mathsf{N}}$ to be a family of sets of trap-door permutations such that the domain of $f \in Tdp_k$ is D_k.

6.1 Standard Definition of a Hard-Core Predicate

Informally, we say that B is a hard-core predicate for a set of trap-door permutations if, given a trap-door permutation f in the set, an element z of the domain of this permutation, and a bit b, no efficient algorithm can guess whether $b = B(f^{-1}(z))$ or is a random bit with a non-negligible advantage.

More precisely, we define a hard-core predicate as follows:

$Src(D, \mu)$:

Signature:

Input:
none

Internal:
$choose - rand$

Output:
$rand(d), d \in D$

State:

$chosenval \in D \cup \{\perp\}$, initially $\perp$

Transitions:

$choose - rand$
Precondition:
$chosenval = \perp$
Effect:
$chosenval :=$ choose-random(D, μ)

$rand(d)$
Precondition:
$d = chosenval \neq \perp$
Effect:
none

Tasks: $\{choose - rand\}$, $\{rand(*)\}$.

Figure 1: Code for $Src(D, \mu)$

Definition 6.1 *A* hard-core predicate *for* $\overline{D}$ *and* $\overline{Tdp}$ *is a predicate* $B : \bigcup_{k \in \mathsf{N}} D_k \rightarrow \{0,1\}$*, such that*

1. *B is polynomial-time computable.*
2. *For every probabilistic polynomial-time non-uniform predicate $G = \{G_k\}_{k \in \mathsf{N}}$,[1] there is a negligible function ϵ such that, for all k,*

$$\left| \begin{array}{ll} \Pr[& f \leftarrow Tdp_k; \\ & z \leftarrow D_k; \\ & b \leftarrow B(f^{-1}(z)) : \\ & G_k(f,z,b) = 1 \quad] \end{array} - \begin{array}{ll} \Pr[& f \leftarrow Tdp_k; \\ & z \leftarrow D_k; \\ & b \leftarrow \{0,1\} : \\ & G_k(f,z,b) = 1 \quad] \end{array} \right| \leq \epsilon(k).$$

Note that, when A is a finite set, the notation $x \leftarrow A$ means that x is selected randomly (according to the uniform distribution) from A.

This definition is a reformulation of Def. 2.5.1 of [Gol01].

6.2 Redefinition of Hard-Core Predicates in Terms of PIOAs

We now show how this last definition can be expressed in terms of task-PIOAs. To this purpose, we define two new task-PIOA families. The first one, denoted by $\overline{SH}$ (for "System providing a Hard-core bit"), outputs a random trap-door permutation, a random element z of the domain of this permutation, and the bit $B(f^{-1}(z))$. The second, denoted by $\overline{SHR}$ (for "System in which the Hard-core bit is replaced by a Random bit"), is the same as the previous one excepted that the output bit b is simply a random bit.

Definition 6.2 *The task-PIOA family* $\overline{SH}$ *is defined as* $hide_{rand(y)_{yval}}(\overline{Src_{tdp}} \| \overline{Src_{yval}} \| \overline{H})$*, where*

- $\overline{Src_{tdp}} = \{(Src_{tdp})_k\}_{k \in \mathsf{N}}$*, where each* $(Src_{tdp})_k$ *is isomorphic to* $Src(Tdp_k)$*,*

[1] This is defined to be a family of predicates that can be evaluated by a non-uniform family $(M_k)_k$ of probabilistic polynomial-time-bounded Turing machines, that is, by a family of Turing machines for which there exist polynomials p and q such that each M_k executes in time at most $p(k)$ and has a standard representation of length at most $q(k)$. An equivalent requirement is that the predicates are computable by a family of Boolean circuits where the k^{th} circuit in the family is of size at most $p(k)$.

- $\overline{Src_{yval}} = \{(Src_{yval})_k\}_{k \in \mathsf{N}}$, *where each* $(Src_{yval})_k$ *is isomorphic to* $Src(D_k)$,
- $\overline{H} = \{H_k\}_{k \in \mathsf{N}}$, *where each* H_k *receives the permutation* f *from* $(Src_{tdp})_k$ *and the element* $y \in D_k$ *from* $(Src_{yval})_k$, *and outputs the two values* $z = f(y)$ *and* $B(y)$. *Each* H_k *is defined as* $H(D_k, Tdp_k, B)$, *where* $H(D, Tdp, B)$ *is defined in Fig. 2.*

$H(D, Tdp, B)$:

Signature:

Input:
- $rand(f)_{tdp}$, $f \in Tdp$
- $rand(y)_{yval}$, $y \in D$

Output:
- $rand(z)_{zval}$, $z \in D$
- $rand(b)_{bval}$, $b \in \{0,1\}$

Internal:
- $fix - bval$
- $fix - zval$

State:
- $fval \in Tdp \cup \perp$, initially $\perp$
- $yval \in D \cup \perp$, initially $\perp$
- $zval \in D \cup \perp$, initially $\perp$
- $bval \in \{0,1\} \cup \perp$, initially $\perp$

Transitions:

$rand(f)_{tdp}$
Effect:
  if $fval = \perp$ then $fval := f$

$rand(y)_{yval}$
Effect:
  if $yval = \perp$ then $yval := y$

$fix - zval$
Precondition:
  $fval \neq \perp$, $yval \neq \perp$
Effect:
  if $zval = \perp$ then $zval := fval(yval)$

$fix - bval$
Precondition:
  $yval \neq \perp$
Effect:
  if $bval = \perp$ then $bval := B(yval)$

$rand(z)_{zval}$
Precondition:
  $z = zval \neq \perp$
Effect:
  none

$rand(b)_{bval}$
Precondition:
  $b = bval \neq \perp$
Effect:
  none

Tasks: $\{fix - bval\}$, $\{fix - zval\}$, $\{rand(*)_{zval}\}$, $\{rand(*)_{bval}\}$.

Figure 2: Hard-core predicate automaton, $H(D, Tdp, B)$

Definition 6.3 *The task-PIOA family* $\overline{SHR}$ *is defined as* $(\overline{Src_{tdp}} \| \overline{Src_{zval}} \| \overline{Src_{bval}})$, *where*

- $\overline{Src_{tdp}} = \{(Src_{tdp})_k\}_{k \in \mathsf{N}}$, *where each* $(Src_{tdp})_k$ *is isomorphic to* $Src(Tdp_k)$,
- $\overline{Src_{zval}} = \{(Src_{zval})_k\}_{k \in \mathsf{N}}$, *where each* $(Src_{zval})_k$ *is isomorphic to* $Src(D_k)$,
- $\overline{Src_{bval}} = \{(Src_{bval})_k\}_{k \in \mathsf{N}}$, *where each* $(Src_{bval})_k$ *is isomorphic to* $Src(\{0,1\})$.

These two systems are represented in Fig. 3. In this figure, the automata labeled with Ⓢ represent the different random source automata. Also, for clarity, the indices of the different actions are removed.

Using these two task-PIOA families, Definition 6.1 of hard-core predicates can be expressed in terms of task-PIOAs by saying that $\overline{SH} \leq_{neg,pt} \overline{SHR}$, which means (informally) that, for every

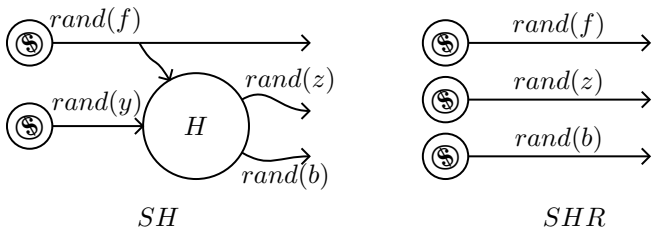

Figure 3: SH and SHR

polynomial-time-bounded family $\overline{\mathcal{E}}$ of environments for $\overline{SH}$ and $\overline{SHR}$, every polynomial-time-bounded task-scheduler family for $\overline{SH}\|\overline{\mathcal{E}}$, generates a family of trace distributions of $\overline{SH}\|\overline{\mathcal{E}}$ that can be mimicked by $\overline{SHR}\|\overline{\mathcal{E}}$ with an appropriate task-scheduler family.

Definition 6.4 *A* hard-core predicate *for $\overline{D}$ and $\overline{Tdp}$ is a polynomial-time-computable predicate $B : \bigcup_{k\in\mathsf{N}} D_k \to \{0,1\}$, such that $\overline{SH} \leq_{neg,pt} \overline{SHR}$.*

We show that this definition is equivalent to Definition 6.1 by means of the following two theorems.

Theorem 6.5 *If B is a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ according to Definition 6.1, then B is also a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ according to Definition 6.4.*

Proof. Suppose that B is a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ according to Definition 6.1. Definition 6.1 implies that B is polynomial-time computable, which is required by Definition 6.4.

It remains to show that $\overline{SH} \leq_{neg,pt} \overline{SHR}$, where the same B defined above is used in the definition of $\overline{SH}$. To show this, we fix polynomials p and p_1. It suffices to show the existence of a negligible function ϵ such that $\overline{SH} \leq_{\epsilon,p,p_1,p_1} \overline{SHR}$. This amounts to proving the existence of a negligible function ϵ such that, for every $k \in \mathsf{N}$, $SH_k \leq_{\epsilon(k),p(k),p_1(k),p_1(k)} SHR_k$. Unwinding this definition further, this means that it is enough to show the existence of a negligible function ϵ such that, for every $k \in \mathsf{N}$, for every $p(k)$-time-bounded environment $\mathcal{E}$ for SH_k and SHR_k, and for every $p_1(k)$-bounded task scheduler ρ_1 for $SH_k\|\mathcal{E}$, there exists a $p_1(k)$-bounded task scheduler ρ_2 for $SHR_k\|\mathcal{E}$, such that

$$|Paccept(SH_k\|\mathcal{E},\rho_1) - Paccept(SHR_k\|\mathcal{E},\rho_2)| \leq \epsilon(k).$$

We first define a homomorphism of task schedulers. Specifically, for every k and every environment $\mathcal{E}$ for SH_k and SHR_k, we define a homomorphism *hom* from task schedulers of $SH_k\|\mathcal{E}$ to task schedulers of $SHR_k\|\mathcal{E}$. Namely,

1. Replace each occurrence of the $\{choose - rand_{yval}\}$ and $\{rand(*)_{yval}\}$ tasks of $(Src_{yval})_k$ with the empty task sequence λ.
2. Replace each occurrence of the $\{fix - bval\}$ task of H_k with the $\{choose - rand_{bval}\}$ task of $(Src_{bval})_k$.
3. Replace each occurrence of the $\{fix - zval\}$ task of H_k with the $\{choose - rand_{zval}\}$ task of $(Src_{zval})_k$.
4. Keep every other task unchanged.

Note that homomorphism *hom* is independent of k and $\mathcal{E}$. Also, note that *hom* is length-nonincreasing: for every task scheduler ρ_1 of $SH_k\|\mathcal{E}$, $|hom(\rho_1)| \leq |\rho_1|$.

Thus, it is enough to show the existence of a negligible function ϵ such that, for every $k \in \mathsf{N}$, for every $p(k)$-time-bounded environment $\mathcal{E}$ for SH_k and SHR_k, and for every $p_1(k)$-bounded task scheduler ρ_1 for $SH_k\|\mathcal{E}$,

$$|Paccept(SH_k\|\mathcal{E},\rho_1) - Paccept(SHR_k\|\mathcal{E},hom(\rho_1))| \leq \epsilon(k).$$

Now, for every $k \in \mathsf{N}$, define $(\mathcal{E}_{max})_k$ to be a $p(k)$-time-bounded environment for SH_k and define $(\rho_{1max})_k$ to be a $p_1(k)$-time-bounded scheduler for $SH_k \| (\mathcal{E}_{max})_k$, with the property that, for every $p(k)$-time-bounded environment $\mathcal{E}$ for SH_k and every $p_1(k)$-time-bounded scheduler ρ_1 for $SH_k \| \mathcal{E}$,

$$|Paccept(SH_k \| \mathcal{E}, \rho_1) - Paccept(SHR_k \| \mathcal{E}, hom(\rho_1))| \leq$$
$$|Paccept(SH_k \| (\mathcal{E}_{max})_k, (\rho_{1max})_k) - Paccept(SHR_k \| (\mathcal{E}_{max})_k, hom((\rho_{1max})_k))|$$

To see that such $(\mathcal{E}_{max})_k$ and $(\rho_{1max})_k$ must exist, note that we are considering only $\mathcal{E}$ for which all parts of the description are bounded by $p(k)$, and only ρ_1 with length at most $p_1(k)$. Since there are only a finite number of such $(\mathcal{E}, \rho_1)$ pairs (up to isomorphism), we can select a particular pair that maximizes the given difference.

This means that it is enough to show the existence of a negligible function ϵ such that, for every $k \in \mathsf{N}$,

$$|Paccept(SH_k \| (\mathcal{E}_{max})_k, (\rho_{1max})_k) - Paccept(SHR_k \| (\mathcal{E}_{max})_k, hom((\rho_{1max})_k))| \leq \epsilon(k).$$

To show this, we will apply Definition 6.1. This requires us to define an appropriate probabilistic polynomial-time non-uniform predicate $G = (G_k)_{k \in \mathsf{N}}$.

We define G_k as follows: G_k has three input arguments: $f \in Tdp_k$, $z \in D_k$ and $b \in \{0,1\}$; we only care what G_k does if its inputs are in these designated sets. For these inputs, G_k simulates the behavior of $(\mathcal{E}_{max})_k$ when it is executed with $(\rho_{1max})_k$, as follows:

1. G_k reads its inputs f, z and b.
2. G_k then reads the tasks in $(\rho_{1max})_k$, one by one. For each task T that it reads:
   - G_k determines (in polynomial time) whether T is one of the following tasks:
     - T is a task of $(\mathcal{E}_{max})_k$,
     - $T \in \{\{rand(*)_{tdp}\}, \{rand(*)_{zval}\}, \{rand(*)_{bval}\}\}$

     and goes on to the next task if it is not.
   - If T is a task of $(\mathcal{E}_{max})_k$, then G_k simulates the performance of T, by determining the unique enabled action (in polynomial time) and the next state (in probabilistic polynomial time).
   - If T is a task of the form $\{rand(*)_{tdp}\}$ then G_k checks if an action of this task is enabled (that is, if the $\{choose - rand_{tdp}\}$-task has already been read in $(\rho_{1max})_k$) and, if it is the case, simulates the action $rand(f)_{tdp}$, where f is G_k's first input argument. If T is disabled, then G_k does nothing. Similarly, if T is of the form $\{rand(*)_{zval}\}$ and if this task is enabled, then G_k simulates the action $rand(z)_{zval}$, where z is G_k's second input argument. And if T is of the form $\{rand(*)_{bval}\}$ and if this task is enabled, then G_k simulates $rand(b)_{bval}$, where b is G_k's third input argument.
3. After completing the processing of $(\rho_{1max})_k$, G_k checks if the *accept* action has been performed. It outputs 1 in that case, and 0 otherwise.

Now, Definition 6.1 guarantees that there is a negligible function ϵ such that, for all k,

$$\left| \begin{array}{lll} \Pr[& f \leftarrow Tdp_k; & \\ & z \leftarrow D_k; & \\ & b \leftarrow B(f^{-1}(z)) : & \\ & G_k(f,z,b) = 1 &] \end{array} - \begin{array}{lll} \Pr[& f \leftarrow Tdp_k; & \\ & z \leftarrow D_k; & \\ & b \leftarrow \{0,1\} : & \\ & G_k(f,z,b) = 1 &] \end{array} \right| \leq \epsilon(k).$$

By the definitions of $\overline{SH}$ and $\overline{SHR}$, and the homomorphism hom, we observe that:

$$Paccept(SH_k \| (\mathcal{E}_{max})_k, (\rho_{1max})_k) = \left(\begin{array}{lll} \Pr[& f \leftarrow Tdp_k; & \\ & z \leftarrow D_k; & \\ & b \leftarrow B(f^{-1}(z)) : & \\ & G_k(f,z,b) = 1 &] \end{array} \right)$$

and

$$Paccept(SHR_k\|(\mathcal{E}_{max})_k, hom((\rho_{1max})_k)) = \left(\begin{array}{lll} \Pr[& f \leftarrow Tdp_k; & \\ & z \leftarrow D_k; & \\ & b \leftarrow \{0,1\} : & \\ & G_k(f,z,b) = 1 &] \end{array} \right).$$

Therefore, we conclude that, for every $k \in \mathsf{N}$,

$$Paccept(SH_k\|(\mathcal{E}_{max})_k, (\rho_{1max})_k) - Paccept(SHR_k\|(\mathcal{E}_{max})_k, hom((\rho_{1max})) \leq \epsilon(k),$$

which is what we needed to show. □

Theorem 6.6 *If B is a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ according to Definition 6.4, then B is also a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ according to Definition 6.1.*

Proof. Suppose that B is a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ according to Definition 6.4. Definition 6.4 implies that B is polynomial-time computable, which is required by Definition 6.1.

It remains to show that, for every probabilistic polynomial-time non-uniform predicate $G = \{G_k\}_{k\in\mathsf{N}}$, there is a negligible function ϵ such that, for every $k \in \mathsf{N}$,

$$\left| \begin{array}{lll} \Pr[& f \leftarrow Tdp_k; & \\ & z \leftarrow D_k; & \\ & b \leftarrow B(f^{-1}(z)) : & \\ & G_k(f,z,b) = 1 &] \end{array} - \begin{array}{lll} \Pr[& f \leftarrow Tdp_k; & \\ & z \leftarrow D_k; & \\ & b \leftarrow \{0,1\} : & \\ & G_k(f,z,b) = 1 &] \end{array} \right| \leq \epsilon(k).$$

For the rest of this proof, we define $PH(G,k)$ and $PHR(G,k)$ as:

$$PH(G,k) = \left(\begin{array}{lll} \Pr[& f \leftarrow Tdp_k; & \\ & z \leftarrow D_k; & \\ & b \leftarrow B(f^{-1}(z)) : & \\ & G_k(f,z,b) = 1 &] \end{array} \right) \text{ and } PHR(G,k) = \left(\begin{array}{lll} \Pr[& f \leftarrow Tdp_k; & \\ & z \leftarrow D_k; & \\ & b \leftarrow \{0,1\} : & \\ & G_k(f,z,b) = 1 &] \end{array} \right)$$

for any non-uniform predicate $G = \{G_k\}_{k\in\mathsf{N}}$ and any $k \in \mathsf{N}$.

Now, fix any probabilistic polynomial-time non-uniform predicate $G = \{G_k\}_{k\in\mathsf{N}}$. Starting from this predicate, we define a polynomial-time-bounded environment family $\overline{\mathcal{E}(G)} = \{(\mathcal{E}(G))_k\}_{k\in\mathsf{N}}$ for both $\overline{SH}$ and $\overline{SHR}$. Each $(\mathcal{E}(G))_k$ is defined as $\mathcal{E}(G_k)(D_k, Tdp_k, B)$, where $\mathcal{E}(G)(D, Tdp, B)$ is defined in Fig. 4.

We also define a polynomial-time-bounded task-scheduler family $\overline{\rho_1} = \{(\rho_1)_k\}_{k\in\mathsf{N}}$ for $\overline{SH}\|\overline{\mathcal{E}(G)}$: for every k,

$$\begin{aligned} (\rho_1)_k = &\{choose - rand_{tdp}\}\{rand(*)_{tdp}\} \\ &\{choose - rand_{yval}\}\{rand(*)_{yval}\} \\ &\{fix - zval\}\{rand(*)_{zval}\} \\ &\{fix - bval\}\{rand(*)_{bval}\} \\ &\{accept\}. \end{aligned}$$

We observe that, from the definition of SH_k, $(\mathcal{E}(G))_k$ and $(\rho_1)_k$:

$$Paccept(SH_k\|(\mathcal{E}(G))_k, (\rho_1)_k) = PH(G,k).$$

Definition 6.4 guarantees that there is a polynomial p and a negligible function ϵ such that, for every k, there is a $p(k)$-bounded task-scheduler $(\rho_2)_k$ such that:

$$|Paccept(SH_k\|(\mathcal{E}(G))_k, (\rho_1)_k) - Paccept(SHR_k\|(\mathcal{E}(G))_k, (\rho_2)_k)| \leq \epsilon(k).$$

$\mathcal{E}(G)(D, Tdp, B)$:

Signature:

Input:
- $rand(f)_{tdp}$, $f \in Tdp$
- $rand(z)_{zval}$, $z \in D$
- $rand(b)_{bval}$, $b \in \{0,1\}$

Output:
- $accept$

State:
- $fval \in Tdp \cup \perp$, initially $\perp$
- $zval \in D \cup \perp$, initially $\perp$
- $bval \in \{0,1\} \cup \perp$, initially $\perp$

Transitions:

$rand(f)_{tdp}$
Effect:
  if $fval = \perp$ then $fval := f$

$rand(z)_{zval}$
Effect:
  if $zval = \perp$ then $zval := z$

$rand(b)_{bval}$
Effect:
  if $bval = \perp$ then $bval := b$

$accept$
Precondition:
  $fval$, $zval$, $bval \neq \perp$
  $G(fval, zval, bval) = 1$
Effect:
  none

Tasks: $\{accept\}$.

Figure 4: Environment evaluating the G predicate, $\mathcal{E}(G)(D, Tdp, B)$

Consider now the probabilistic polynomial-time non-uniform predicate $G' = \{G'_k\}_{k \in \mathsf{N}}$ where $G'_k = 1 - G_k$. For this predicate, Def. 6.4 also guarantees that there are a polynomial p' and a negligible function ϵ' such that, for every k, there is a $p'(k)$-bounded task-scheduler $(\rho'_2)_k$ such that:

$$|Paccept(SH_k \| (\mathcal{E}(G'))_k, (\rho_1)_k) - Paccept(SHR_k \| (\mathcal{E}(G'))_k, (\rho'_2)_k)| \leq \epsilon'(k).$$

We now define a new negligible function ϵ_{max} as $\epsilon_{max}(k) = \max(\epsilon(k), \epsilon'(k))$ for every k. Since ϵ_{max} is a negligible function, there is an index k_0 such that $\epsilon_{max}(k) < \frac{1}{2}$ for every $k \geq k_0$. Let us examine the possible values of $Paccept(SHR_k \| (\mathcal{E}(G))_k, (\rho_2)_k)$ for every $k \geq k_0$.

Fix any $k \geq k_0$. Suppose first that $(\rho_2)_k$ is such that:

$$Paccept(SHR_k \| (\mathcal{E}(G))_k, (\rho_2)_k) = PHR(G, k),$$

which is the case when $(\rho_2)_k$ schedules the $choose - rand_{tdp}$ task followed by the $rand(*)_{tdp}$ task, the $choose - rand_{zval}$ task followed by the $rand(*)_{zval}$ task, the $choose - rand_{bval}$ task followed by the $rand(*)_{bval}$ task, and all these tasks followed by the $accept$ task (in the rest of this proof, we will refer to this as $(\rho_2)_k$ correctly scheduling $accept$). For this $(\rho_2)_k$, we have that

$$|PH(G, k) - PHR(G, k)| \leq \epsilon(k) \leq \epsilon_{max}(k).$$

Suppose now that $(\rho_2)_k$ is such that $Paccept(SHR_k \| (\mathcal{E}(G))_k, (\rho_2)_k)$ is independent of G, that is, $(\rho_2)_k$ does not correctly schedule the $accept$ task. In that case, $Paccept(SHR_k \| (\mathcal{E}(G))_k, (\rho_2)_k) = 0$. Therefore,

$$Paccept(SH_k \| (\mathcal{E}(G))_k, (\rho_1)_k) \leq \epsilon(k) \leq \epsilon_{max}(k) < \frac{1}{2}$$

and

$$Paccept(SH_k \| (\mathcal{E}(G'))_k, (\rho_1)_k) = 1 - Paccept(SH_k \| (\mathcal{E}(G))_k, (\rho_1)_k) > \frac{1}{2},$$

which in turn imply that $Paccept(SHR_k \| (\mathcal{E}(G'))_k, (\rho_2')_k) > 0$ since $\epsilon'(k) < \frac{1}{2}$. But the probability $Paccept(SHR_k \| (\mathcal{E}(G'))_k, (\rho_2')_k)$ can only be different from 0 if $(\rho_2')_k$ correctly schedules *accept*. So, we have that:

$$Paccept(SHR_k \| (\mathcal{E}(G'))_k, (\rho_2')_k) = PHR(G', k)$$

and

$$\begin{aligned} |PH(G,k) - PHR(G,k)| &= |(1 - PH(G',k)) - (1 - PHR(G',k))| \\ &= |PH(G',k) - PHR(G',k)| \\ &\leq \epsilon'(k) \leq \epsilon_{max}(k). \end{aligned}$$

So, $|PH(G,k) - PHR(G,k)| \leq \epsilon_{max}(k)$ for every $k \geq k_0$. Finally, if we define the negligible function ϵ'_{max} as:

$$\epsilon'_{max}(k) = \begin{cases} 1 & \text{if } k < k_0 \\ \epsilon_{max}(k) & \text{otherwise,} \end{cases}$$

the relation $|PH(G,k) - PHR(G,k)| \leq \epsilon'_{max}(k)$ holds for every $k \in \mathsf{N}$, as needed. □

6.3 Consequences of the New Definition

In this subsection, we provide a first illustration of the way our definition of hard-core predicates can be exploited in the analysis of more complicate systems. More precisely, we show that a hard-core predicate can be applied to two values, and a probabilistic polynomial-time environment still cannot distinguish the results from random values. This fact is needed because, in the Oblivious Transfer protocol we analyze, the transmitter applies the hard-core predicate to both $f^{-1}(zval(0))$ and $f^{-1}(zval(1))$, where f is the chosen trap-door function.

Now, we show, if B is a hard-core predicate, then no probabilistic polynomial-time environment can distinguish the distribution $(f, z(0), z(1), B(f^{-1}(z(0))), B(f^{-1}(z(1))))$ from the distribution $(f, z(0), z(1), b(0), b(1))$, where f is a randomly-chosen trap-door permutation, $z(0)$ and $z(1)$ are randomly-chosen elements of the domain D_k, and $b(0)$ and $b(1)$ are randomly-chosen bits. We do this by defining two systems that produce the two distributions, and showing that one implements the other. We use our second definition of hard-core predicate, Definition 6.4.

Definition 6.7 *The task-PIOA family* $\overline{SH2}$ *is defined as* $hide(\overline{Src_{tdp}} \| \overline{Src_{yval0}} \| \overline{Src_{yval1}} \| \overline{H0} \| \overline{H1}, \{rand(*)_{yval0}, rand(*)_{yval1}\})$, *where*

- $\overline{Src_{tdp}} = \{(Src_{tdp})_k\}_{k \in \mathsf{N}}$, *where each* $(Src_{tdp})_k$ *is isomorphic to* $Src(Tdp_k)$,
- $\overline{Src_{yval0}} = \{(Src_{yval0})_k\}_{k \in \mathsf{N}}$, $\overline{Src_{yval1}} = \{(Src_{yval1})_k\}_{k \in \mathsf{N}}$, *where each* $(Src_{yval0})_k$ *and each* $(Src_{yval1})_k$ *is isomorphic to* $Src(D_k)$,
- $\overline{H0} = \{H0_k\}_{k \in \mathsf{N}}$ *and* $\overline{H1} = \{H1_k\}_{k \in \mathsf{N}}$ *are two instances of* $\overline{H}$, *where all actions have the corresponding index* 0 *or* 1 *appended to their name (e.g.,* $rand(z)_{zval}$ *is renamed as* $rand(z)_{zval0}$ *in* $\overline{H0}$*). The only exception is the* $rand(f)_{tdp}$ *action, which is kept as it is in* $\overline{H}$*: we use the same trapdoor permutation for both task-PIOA families.*

Definition 6.8 *The task-PIOA family* $\overline{SHR2}$ *is defined as* $(\overline{Src_{tdp}} \| \overline{Src_{zval0}} \| \overline{Src_{zval1}} \| \overline{Src_{bval0}} \| \overline{Src_{bval1}})$, *where*

- $\overline{Src_{tdp}} = \{(Src_{tdp})_k\}_{k \in \mathsf{N}}$, *where each* $(Src_{tdp})_k$ *is isomorphic to* $Src(Tdp_k)$,
- $\overline{Src_{zval0}} = \{(Src_{zval0})_k\}_{k \in \mathsf{N}}$ *and* $\overline{Src_{zval1}} = \{(Src_{zval1})_k\}_{k \in \mathsf{N}}$, *where each* $(Src_{zval0})_k$ *and each* $(Src_{zval1})_k$ *is isomorphic to* $Src(D_k)$,

- $\overline{Src_{bval0}} = \{(Src_{bval0})_k\}_{k\in\mathsf{N}}$ *and* $\overline{Src_{bval1}} = \{(Src_{bval1})_k\}_{k\in\mathsf{N}}$*, where each* $(Src_{bval0})_k$ *and each* $(Src_{bval1})_k$ *is isomorphic to* $Src(\{0,1\})$

Lemma 6.9 *If B is a hard-core predicate, then $\overline{SH2} \leq_{neg,pt} \overline{SHR2}$.*

Proof. By Theorem 6.5, we may assume that $\overline{SH} \leq_{neg,pt} \overline{SHR}$. To prove that $\overline{SH2} \leq_{neg,pt} \overline{SHR2}$, we introduce a new task-PIOA family $\overline{Int}$, which is intermediate between $\overline{SH2}$ and $\overline{SHR2}$. $\overline{Int}$ is defined as $hide(\overline{Src_{tdp}}\|\overline{Src_{yval0}}\|\overline{H0}\|\overline{Src_{zval1}}\|\overline{Src_{bval1}}, \{rand(*)_{yval0}\})$, where

- $\overline{Src_{tdp}}$ is exactly as in $\overline{SH2}$ and $\overline{SHR2}$.
- $\overline{Src_{yval0}}$ and $\overline{H0}$ are as in $\overline{SH2}$.
- $\overline{Src_{zval1}}$ and $\overline{Src_{bval1}}$ are as in $\overline{SHR2}$.

Thus, $\overline{Int}$ generates one of the bits, $bval0$, using the hard-core predicate B, as in $\overline{SH2}$, and generates the other, $bval1$, randomly, as in $\overline{SHR2}$.

We claim that $\overline{SH2} \leq_{neg,pt} \overline{Int}$. To see this, note that Definition 6.1 implies that

$$hide(\overline{Src_{tdp}}\|\overline{Src_{yval1}}\|\overline{H1}, \{rand(*)_{yval1}\}) \leq_{neg,pt} \overline{Src_{tdp}}\|\overline{Src_{zval1}}\|\overline{Src_{bval1}}.$$

This is because these two systems are simple renamings of the $\overline{SH}$ and $\overline{SHR}$ systems described in Section 6.2.

Now let $\overline{I}$ be the task-PIOA family $hide_{rand(y)_{yval0}}(\overline{Src}_{yval0}\|\overline{H0})$. It is easy to see, from the code for the two components of $\overline{I}$, that $\overline{I}$ is polynomial-time-bounded. Then Lemma 4.19 implies that

$$hide(\overline{Src_{tdp}}\|\overline{Src_{yval1}}\|\overline{H1}, \{rand(*)_{yval1}\})\|\overline{I} \leq_{neg,pt} \overline{Src_{tdp}}\|\overline{Src_{zval1}}\|\overline{Src_{bval1}}\|\overline{I}.$$

Since the left-hand side of this relation is $\overline{SH2}$ and the right-hand side is $\overline{Int}$, this implies $\overline{SH2} \leq_{neg,pt} \overline{Int}$, as needed.

Next, we claim that $\overline{Int} \leq_{neg,pt} \overline{SHR2}$. To see this, note that Definition 6.1 implies that

$$hide(\overline{Src_{tdp}}\|\overline{Src_{yval0}}\|\overline{H0}, \{rand(*)_{yval0}\}) \leq_{neg,pt} \overline{Src_{tdp}}\|\overline{Src_{zval0}}\|\overline{Src_{bval0}}.$$

Now let $\overline{I}$ be the polynomial-time-bounded task-PIOA family $\overline{Src_{zval1}}\|\overline{Src_{bval1}}$. Then Lemma 4.19 implies that

$$hide(\overline{Src_{tdp}}\|\overline{Src_{yval0}}\|\overline{H0}, \{rand(*)_{yval0}\})\|\overline{I} \leq_{neg,pt} \overline{Src_{tdp}}\|\overline{Src_{zval0}}\|\overline{Src_{bval0}}\|\overline{I}.$$

Since the left-hand side of this relation is $\overline{Int}$ and the right-hand side is $\overline{SHR2}$, this implies $\overline{Int} \leq_{neg,pt} \overline{SHR2}$, as needed.

Since $\overline{SH2} \leq_{neg,pt} \overline{Int}$ and $\overline{Int} \leq_{neg,pt} \overline{SHR2}$, transitivity of $\leq_{neg,pt}$ (Lemma 4.28) implies that $\overline{SH2} \leq_{neg,pt} \overline{SHR2}$. □

7 Specification for Oblivious Transfer

In this section, we define "ideal systems" for Oblivious Transfer, which are used as specifications for the correctness and secrecy properties that are supposed to be guaranteed by an Oblivious Transfer protocol.

We parameterize our ideal systems by a set $C \subseteq \{Trans, Rec\}$, which indicates the corrupted end-points.

The system, represented in Figure 5 consists of two interacting task-PIOAs: the Functionality *Funct* and the Simulator *Sim*, which we describe now.

Notation: The states of each task-PIOA for which we provide explicit code are structured in terms of a collection of state variables. Given a state q of a task-PIOA and a state variable v, we write $q.v$ for the value of v in state q.

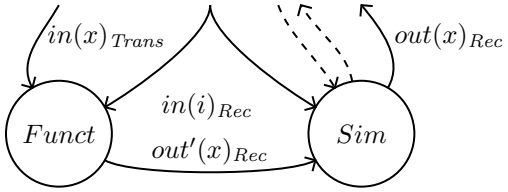

Figure 5: $Funct$ and Sim

7.1 The Oblivious Transfer Functionality

The complete definition of *Funct* is given in Figure 6. It has two inputs corresponding to the inputs sent by the environment to *Trans* and *Rec*, and one output that transmits the output intended to the receiver *Rec*. The definitions are based on Canetti's definition of Oblivious Transfer in the Universal Composability framework [Can01].

$Funct(C)$:

Signature:

Input:
- $in(x)_{Trans}$, $x \in (\{0,1\} \to \{0,1\})$
- $in(i)_{Rec}$, $i \in \{0,1\}$

Output:
- if $Rec \notin C$ then $out(x)_{Rec}$, $x \in \{0,1\}$
- if $Rec \in C$ then $out'(x)_{Rec}$, $x \in \{0,1\}$

State:

- $inval(Trans) \in (\{0,1\} \to \{0,1\}) \cup \{\perp\}$, initially $\perp$
- $inval(Rec) \in \{0,1,\perp\}$, initially $\perp$

Transitions:

$in(x)_{Trans}$
Effect:
- if $inval(Trans) = \perp$ then $inval(Trans) := x$

$in(i)_{Rec}$
Effect:
- if $inval(Rec) = \perp$ then $inval(Rec) := i$

$out(x)_{Rec}$ or $out'(x)_{Rec}$
Precondition:
- $inval(Trans), inval(Rec) \neq \perp$
- $x = inval(Trans)(inval(Rec))$

Effect:
- none

Tasks:

If $Rec \notin C$ then $\{out(*)_{Rec}\}$.
If $Rec \in C$ then $\{out'(*)_{Rec}\}$.

Figure 6: The Functionality, $Funct(C)$

7.2 The Simulator

Simulator $Sim(C)$ is defined as an arbitrary task-PIOA satisfying certain constraints regarding its signature. This is why the composition of *Funct* and *Sim* in fact specifies a class of systems rather than a single one. $Sim(C)$ receives *in* inputs at endpoints in C. It also acts as an intermediary for outputs

at *Rec* if $Rec \in C$, receiving out' outputs from $Funct(C)$ and producing *out* outputs. $Sim(C)$ may also have other, arbitrary, input and output actions. The constraints on the signature and relations of $Sim(C)$ is given in Figure 7.

Signature:

Input:
- if $Trans \in C$ then
  - $in(x)_{Trans}$, $x \in (\{0,1\} \to \{0,1\})$
- if $Rec \in C$ then
  - $in(i)_{Rec}$, $i \in \{0,1\}$
  - $out'(x)_{Rec}$, $x \in \{0,1\}$
- Arbitrary other input actions

Output:
- if $Rec \in C$ then
  - $out(x)_{Rec}$, $x \in \{0,1\}$
- Arbitrary other output actions

Internal:
- Arbitrary internal actions

Tasks:

If $Rec \in C$ then $\{out(*)_{Rec}\}$.
Arbitrary tasks for other actions.

Figure 7: Constraints on $Sim(C)$

7.3 The Complete System

A complete *ideal system* with parameter C is obtained by composing the task-PIOA $Funct(C)$ with some $Sim(C)$, and then, if $Rec \in C$, hiding the $\{out'(*)_{Rec}\}$ task.

8 Real Systems

A real system is defined as a parameterized task-PIOA, with the following parameters:

- D, a finite domain.
- Tdp, a set of trap door permutations for domain D.
- $C \subseteq \{Trans, Rec\}$, representing the corrupted endpoints.

Based on these, we define the following derived sets:

- $Tdpp = \{(f, f^{-1}) : f \in Tdp\}$, the set of trap door permutation pairs for domain D. If $p = (f, f^{-1}) \in Tdpp$, then we refer to the components f and f^{-1} of p using record notation, as $p.funct$ and $p.inverse$, respectively.
- M, the message alphabet, equal to $\{(1, f) : f \in Tdp\} \cup \{(2, z) : z \in (\{0,1\} \to D)\} \cup \{(3, b) : b \in (\{0,1\} \to \{0,1\})\}$.

A real system with parameters (D, Tdp, C) consists of five interacting task-PIOAs: The Transmitter $Trans(D, Tdp)$, the Receiver $Rec(D, Tdp, C)$, the Adversary $Adv(D, Tdp, C)$, and two random source automata $Src(Tdpp)_{tdpp}$ and $Src(\{0,1\} \to D)_{yval}$. $Src(Tdpp)_{tdpp}$ and $Src(\{0,1\} \to D)_{yval}$ are isomorphic to $Src(Tdpp)$ and $Src(\{0,1\} \to D)$ defined as in Section 5; the difference is that the literal subscripts *tdpp* and *yval* are added to the names of the automata and to their actions. Throughout this section, we abbreviate the automaton names by omiting their parameters when no confusion seems likely.

8.1 The Transmitter

$Trans(D, Tdp)$ receives *in* inputs from the environment of the real system. It produces *send* outputs to and receives *receive* inputs from *Adv*. It also receives $rand_{tdpp}$ inputs from Src_{tdpp}. Task-PIOA $Trans(D, Tdp)$ is defined in Figure 8.

$Trans(D, Tdp)$:

Signature:

Input:
- $in(x)_{Trans}$, $x \in (\{0,1\} \to \{0,1\})$
- $rand(p)_{tdpp}$, $p \in Tdpp$
- $receive(2,z)_{Trans}$, $z \in (\{0,1\} \to D)$

Output:
- $send(1,f)_{Trans}$, $f \in Tdp$
- $send(3,b)_{Trans}$, $b \in (\{0,1\} \to \{0,1\})$

Internal:
- $fix - bval_{Trans}$

State:
- $inval \in (\{0,1\} \to \{0,1\}) \cup \{\perp\}$, initially $\perp$
- $tdpp \in Tdpp \cup \{\perp\}$, initially $\perp$
- $zval \in (\{0,1\} \to D) \cup \{\perp\}$, initially $\perp$
- $bval \in (\{0,1\} \to \{0,1\}) \cup \{\perp\}$, initially $\perp$

Transitions:

$in(x)_{Trans}$
Effect:
  if $inval = \perp$ then $inval := x$

$rand(p)_{tdpp}$
Effect:
  if $tdpp = \perp$ then $tdpp := p$

$send(1,f)_{Trans}$
Precondition:
  $tdpp \neq \perp$, $f = tdpp.funct$
Effect:
  none

$receive(2,z)_{Trans}$
Effect:
  if $zval = \perp$ then $zval := z$

$fix - bval_{Trans}$
Precondition:
  $tdpp, zval, inval \neq \perp$
  $bval = \perp$
Effect:
  for $i \in \{0,1\}$ do
    $bval(i) = B(tdpp.inverse(zval(i))) \oplus inval(i)$

$send(3,b)_{Trans}$
Precondition:
  $b = bval \neq \perp$
Effect:
  none

Tasks: $\{send(1,*)_{Trans}\}$, $\{send(3,*)_{Trans}\}$, $\{fix - bval_{Trans}\}$.

Figure 8: Code for $Trans(D, Tdp)$

Lemma 8.1 *In every reachable state of $Trans(D, Tdp)$: If $bval \neq \perp$ then $tdpp \neq \perp$, $zval \neq \perp$, $inval \neq \perp$, and $\forall i \in \{0,1\}, bval(i) = B(tdpp.inverse(zval(i))) \oplus inval(i)$.*

8.2 The Receiver

$Rec(D, Tdp, C)$ receives *in* inputs from the environment of the real system. Also, if $Rec \in C$, then $Rec(D, Tdp, C)$ produces out' outputs to Adv, whereas if $Rec \notin C$, then $Rec(D, Tdp, C)$ produces *out* outputs for the environment. $Rec(D, Tdp, C)$ provides *send* outputs to and receives *receive* inputs from Adv. It also receives $rand_{yval}$ inputs from Src_{yval}.

Task-PIOA $Rec(D, Tdp, C)$ is defined in Figure 9.

Lemma 8.2 *In every reachable state of $Rec(D, Tdp, C)$:*

1. *If $zval = z \neq \perp$ then $yval \neq \perp$, $inval \neq \perp$, $tdp \neq \perp$, $z(inval) = tdp(yval(inval))$, and $z(1 - inval) = yval(1 - inval)$.*

8.3 The Adversary

The Adversary encompasses the communication channel, although its powers to affect the communication are weak (it can hear messages and decide when to deliver them, but cannot manufacture or

$Rec(D, Tdp, C)$:

Signature:

Input:
- $in(i)_{Rec}$, $i \in \{0,1\}$
- $rand(y)_{yval}, y \in (\{0,1\} \to D)$
- $receive(1,f)_{Rec}$, $f \in Tdp$
- $receive(3,b)_{Rec}$, $b \in (\{0,1\} \to \{0,1\})$

Output:
- $send(2,z)_{Rec}$, $z \in (\{0,1\} \to D)$
- if $Rec \notin C$ then $out(x)_{Rec}$, $x \in \{0,1\}$
- if $Rec \in C$ then $out'(x)_{Rec}$, $x \in \{0,1\}$

Internal:
- $fix - zval_{Rec}$

State:
- $inval \in \{0,1,\perp\}$, initially $\perp$
- $tdp \in Tdp \cup \{\perp\}$, initially $\perp$
- $yval, zval \in (\{0,1\} \to D) \cup \{\perp\}$, initially $\perp$
- $outval \in \{0,1,\perp\}$, initially $\perp$

Transitions:

$in(i)_{Rec}$
Effect:
- if $inval = \perp$ then $inval := i$

$rand(y)_{yval}$
Effect:
- if $yval = \perp$ then $yval := y$

$receive(1,f)_{Rec}$
Effect:
- if $tdp = \perp$ then $tdp := f$

$fix - zval_{Rec}$
Precondition:
- $yval, inval, tdp \neq \perp$
- $zval = \perp$

Effect:
- $zval(inval) := tdp(yval(inval))$
- $zval(1 - inval) := yval(1 - inval)$

$send(2,z)_{Rec}$
Precondition:
- $z = zval \neq \perp$

Effect:
- none

$receive(3,b)_{Rec}$
Effect:
- if $yval \neq \perp$ and $outval = \perp$ then
  - $outval := b(inval) \oplus B(yval(inval))$

$out(x)_{Rec}$ or $out'(x)_{Rec}$
Precondition:
- $x = outval \neq \perp$

Effect:
- none

Tasks:

$\{send(2,*)_{Rec}\}$, $\{fix - zval_{Rec}\}$.
If $Rec \in C$ then $\{out'(*)_{Rec}\}$.
If $Rec \notin C$ then $\{out(*)_{Rec}\}$.

Figure 9: Code for $Rec(D, Tdp, C)$

corrupt messages).

$Adv(D, Tdp, C)$ has two endpoints corresponding to *Trans* and *Rec*. It receives *in* inputs from the environment for endpoints in C. It also acts as an intermediary for outputs at endpoints in C, specifically, if $R \in C$, $Adv(D, Tdp, C)$ receives out' outputs from *Rec* and provides *out* outputs to the environment at endpoint *Rec*. $Adv(D, Tdp, C)$ also receives *send* inputs from and provides *receive* outputs to *Trans* and *Rec*. It also receives random inputs from the random sources of corrupted parties: $rand(p)_{tdpp}$ from Src_{tdpp} if *Trans* $\in C$ and $rand(y)_{yval}$ if $Rec \in C$. Finally, $Adv(D, Tdp, C)$ may communicate with the environment, using other, arbitrary inputs and outputs. We call these "new" inputs and outputs here. We assume that they are disjoint from all the other actions that appear in any of our explicitly-defined components. Thus, they will not be shared with any other state components we define. (Later, when we consider closing the system with an environment automaton, we will allow these new actions to be shared with the environment.)

The Adversary again depends on the set C of corrupted parties. Also, for each case, there are actually a set of possible adversary automata, not just one. This set is captured by the "arbitrary" designation throughout the descriptions. The Adversary $Adv(D, Tdp, C)$ is defined in Figures 10 and 11.

$Adv(D, Tdp, C)$:

Signature:

Input:
- $send(1, f)_{Trans}$, $f \in Tdp$
- $send(2, z)_{Rec}$, $z \in (\{0,1\} \to D)$
- $send(3, b)_{Trans}$, $b \in (\{0,1\} \to \{0,1\})$
- if $T \in C$ then
  - $in(x)_{Trans}$, $x \in (\{0,1\} \to \{0,1\})$
  - $rand(p)_{tdpp}$, $p \in Tdpp$
- if $Rec \in C$ then $in(i)_{Rec}$, $i \in \{0,1\}$
  - $out'(x)_{Rec}$, $x \in \{0,1\}$
  - $rand(y)_{yval}$, $y \in (\{0,1\} \to D)$
- Arbitrary other input actions; call these "new" input actions

Output:
- $receive(1, f)_{Rec}$, $f \in Tdp$
- $receive(2, z)_{Trans}$, $z \in (\{0,1\} \to D)$
- $receive(3, b)_{Rec}$, $b \in (\{0,1\} \to \{0,1\})$
- if $R \in C$ then
  - $out(x)_{Rec}$, $x \in \{0,1\}$
- Arbitrary other output actions,
  - call these "new" output actions

Internal:
- Arbitrary internal actions;
  - call these "new" internal actions

State:
- $messages$, a set of pairs in $M \times \{Trans, Rec\}$, initially $\emptyset$
- if $R \in C$ then $outval(Rec) \in \{0, 1, \perp\}$, initially $\perp$
- Arbitrary other variables; call these "new" variables

Transitions:

$send(m)_{Trans}$
Effect:
- $messages := messages \cup \{(m, Rec)\}$

$send(m)_{Rec}$
Effect:
- $messages := messages \cup \{(m, Trans)\}$

$receive(m)_{Trans}$
Precondition:
- $(m, Trans) \in messages$

Effect:
- none

$receive(m)_{Rec}$
Precondition:
- $(m, Rec) \in messages$

Effect:
- none

$out'(x)_{Rec}$
Effect:
- if $outval(Rec) = \perp$ then $outval(Rec) := x$

$out(x)_{Rec}$
Precondition:
- $x = outval(Rec) \neq \perp$

Effect:
- none

$in(x)_{Trans}$, $in(i)_{Rec}$, $rand(p)_{tdpp}$, or $rand(y)_{yval}$
Effect:
- Arbitrary changes to new state variables

New input action
Effect:
- Arbitrary changes to new state variables

New output or internal action
Precondition:
- Arbitrary

Effect:
- Arbitrary changes to new state variables

Figure 10: Code for $Adv(D, Tdp, C)$ (Part I)

Tasks: $\{receive(1, *)_{Rec}\}$, $\{receive(2, *)_{Trans}\}$, $\{receive(3, *)_{Rec}\}$,
If $Rec \in C$ then $\{out(*)_{Rec}\}$.
Arbitrary tasks for new actions.

Figure 11: Code for $Adv(D, Tdp, C)$ (Part II)

8.4 The complete system

A complete *real system* with parameters (D, Tdp, C) is the result of composing the task-PIOAs $Trans(D, Tdp)$, $Rec(D, Tdp, C)$, $Src(Tdpp)_{tdpp}$ and $Src(\{0,1\} \rightarrow D)_{yval}$ and some adversary $Adv(D, Tdp, C)$, and then, hiding all the *send*, *receive* and *rand* actions. If $Rec \in C$ we also hide out' outputs of *Rec*.

For instance, Figure 12 shows the interconnections of the parties that make up the real system when *Rec* is corrupted. In this figure, we abbreviate all *send* and *receive* actions with the s and r letters respectively. The T and R subscript refer to *Trans* and *Rec*. The dashed arrows represent the arbitrary actions shared by the Adversary and the Environment. The automata labeled with $ represent the random source automata $Src(Tdpp)_{tdpp}$ and $Src(\{0,1\} \rightarrow D)_{yval}$.

Invariants can be proved for this system.

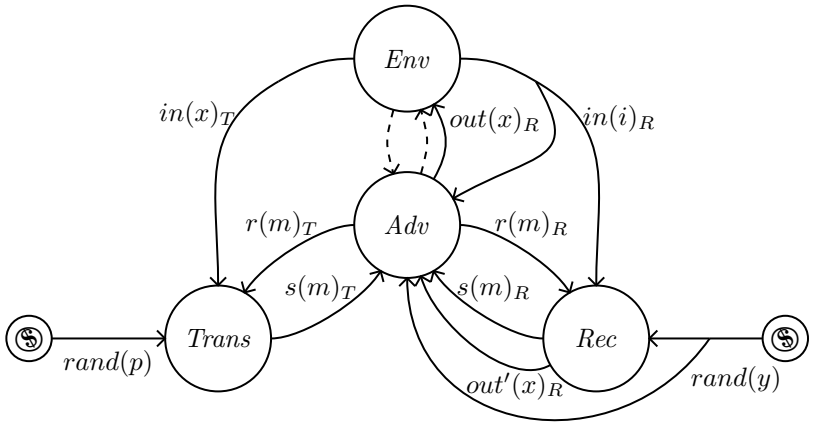

Figure 12: RS when $C = \{Rec\}$

Lemma 8.3 *In every reachable state of RS the following hold:*

1. *Adv.messages contains at most one round 1 message, at most one round 2 message, and at most one round 3 message.*
2. *If Adv.messages contains* $(1, f)$ *then* $Trans.tdpp.funct = f$.
3. *If Adv.messages contains* $(2, z)$ *then* $Rec.zval = z$.
4. *If Adv.messages contains* $(3, b)$ *then* $Trans(D, Tdp).bval = b$.
5. *If* $Rec.tdp = f \neq \perp$ *then*
   (a) *Adv.messages contains* $(1, f)$.
   (b) $Trans.tdpp \neq \perp$ *and* $Trans.tdpp.funct = f$.
6. *If* $Rec.zval = z \neq \perp$ *then* $Rec.yval \neq \perp$, $Rec.inval \neq \perp$, $Rec.tdp \neq \perp$, $z(Rec.inval) = Rec.tdp(Rec.yval(Rec.inval))$, *and* $z(1 - Rec.inval) = Rec.yval(1 - Rec.inval)$.
7. *If* $Trans.zval = z \neq \perp$ *then*
   (a) *Adv.messages contains* $(2, z)$.
   (b) $Rec.zval = z$.
8. *If* $Trans.bval = b \neq \perp$ *then*

(a) $Trans.tdpp \neq \perp$, $Trans.zval \neq \perp$, $Trans.inval \neq \perp$, and $i \in \{0,1\}$, $b(i) = B(Trans.tdpp.inverse(Trans.zval(i))) \oplus Trans.inval(i)$.

(b) $Rec.inval \neq \perp$ and for $i = Rec.inval$, $b(i) = B(Rec.yval(i)) \oplus Trans.inval(i)$.

9. If $Rec.outval = x \neq \perp$ then

(a) $x = Trans.bval(Rec.inval) \oplus B(Rec.yval(Rec.inval))$.

(b) $x = Trans.inval(Rec.inval)$.

10. If $Trans.tdpp \neq \perp$ and $Trans.zval \neq \perp$, then $Rec.yval \neq \perp$, $Rec.inval \neq \perp$, and in addition $Trans.tdpp.inverse(Trans.zval(Rec.inval)) = Rec.yval(Rec.inval)$.

11. If $Rec.yval \neq \perp$ then $Src_{yval}.chosenval = Rec.yval$.

12. If $Trans.tdpp \neq \perp$ then $Src_{tdpp}.chosenval = Trans.tdpp$.

In addition, invariants can be proved for the four individual cases, for instance:

Lemma 8.4 *If $C = \{Rec\}$ then, in every reachable state of $RS(D, Tdp, C)$, the following holds:*

1. *If $Adv.outval(Rec) = b \neq \perp$ then $Rec.outval = b$.*

9 The Main Theorem

In this section, we state the main theorem of this paper.

The theorem involves task-PIOA families, which are defined by instantiating the real and ideal systems with families of domains and trap-door permutations.

9.1 Families of Sets

Using the two families of sets $\overline{D} = \{D_k\}_{k \in \mathbf{N}}$ and $\overline{Tdp} = \{Tdp_k\}_{k \in \mathbf{N}}$ we defined in Section 6, we define the following derived families of sets:

- $\overline{Tdpp} = \{Tdpp_k\}_{k \in \mathbf{N}}$, a family of sets of trap-door permutations pairs. Each set $Tdpp_k$ is the set $\{(f, f^{-1}) : f \in Tdp_k\}$. As before, if $p = (f, f^{-1})$ then we refer to the two components of p as $p.funct$ and $p.inverse$, respectively.
- $\overline{M} = \{M_k\}_{k \in \mathbf{N}}$, a family of message alphabets, where $M_k = \{(1, f) : f \in Tdp_k\} \cup \{(2, z) : z \in (\{0,1\} \rightarrow D_k)\} \cup \{(3, b) : b \in (\{0,1\} \rightarrow \{0,1\})\}$.

9.2 Families of Systems

A *real-system family* $\overline{RS}$ for domain family $\overline{D}$, trap-door permutation set family $\overline{Tdp}$, and $C \subseteq \{Trans, Rec\}$ is a family $\{RS_k\}_{k \in \mathbf{N}}$, where, for each k, RS_k is a real system with parameters (D_k, Tdp_k, C). Thus, $RS_k = Trans(D_k, Tdp_k) \| Rec(D_k, Tdp_k, C) \| Src(Tdpp_k)_{tdpp} \| Src(\{0,1\} \rightarrow D_k)_{yval} \| Adv_k$, where Adv_k is some adversary $Adv(D_k, Tdp_k, C)$.

An *ideal-system family* $\overline{IS}$ for $C \subseteq \{Trans, Rec\}$ is a family $\{IS_k\}_{k \in \mathbf{N}}$, where, for each k, IS_k is an ideal system with parameter C. Thus, $IS_k = Funct(C)_k \| Sim_k$, where Sim_k is some simulator $Sim(C)$.

9.3 Theorem Statement

Theorem 9.1 *For every $C = \{Rec\}$ the following holds:*
Let $\overline{RS}$ be a real-system family for $(\overline{D}, \overline{Tdp}, C)$, in which the family $\overline{Adv}$ of adversary automata is polynomial-time-bounded.
Then there exists an ideal-system family $\overline{IS}$ for C, in which the family $\overline{Sim}$ is polynomial-time-bounded, and such that $\overline{RS} \leq_{neg,pt} \overline{IS}$.

Note that in [CCK+05], the above theorem was stated for every $C \subseteq \{Trans, Rec\}$, hence covering all possible values for the set of corrupted parties. We believe that we can prove that theorem using our generalized notion of task-PIOAs, which is the one defined in this paper and our new notion of simulation relation. We foresee that minor modifications to the formulation of the ideal system will be necessary to achieve the "synchronization" between the actions of the ideal system and the real system in showing the simulation relation.

10 Correctness proof

This section contains the most interesting case: where only the receiver is corrupted. We prove the following theorem:

Theorem 10.1 *Let $\overline{RS}$ be a real-system family for $(\overline{D}, \overline{Tdp}, C)$, $C = \{Rec\}$, in which the family $\overline{Adv}$ of adversary automata is polynomial-time-bounded.*
Then there exists an ideal-system family $\overline{IS}$ for $C = \{Rec\}$, in which the family $\overline{Sim}$ is polynomial-time-bounded, and such that $\overline{RS} \leq_{neg,pt} \overline{IS}$.

Since $C = \{Rec\}$ everywhere in this section, we drop explicit mention of C.

We express each Sim_k as a composition of automata. This composition describes the particular simulation strategy needed to mimic the behavior of the real system. We define a "structured ideal system" SIS_k to be the composition of a structured simulator $SSim_k$ with $Funct_k$. It is easy to see that $\overline{SIS}$ is an ideal-system family, according to our definition of an ideal system. Moreover, if $\overline{Adv}$ is polynomial-time-bounded, then $\overline{Sim}$ is also polynomial-time-bounded. It remains to show that $\overline{RS} \leq_{neg,pt} \overline{SIS}$.

In order to show that $\overline{RS} \leq_{neg,pt} \overline{SIS}$, we use two intermediate families of systems, $\overline{Int1}$ and $\overline{Int2}$. These two families of systems are nearly identical; in fact, they differ only in that $\overline{Int1}$ uses a hard-core predicate of a trap-door permutation in situations where $\overline{Int2}$ uses random bits. Then the proof breaks down into three pieces, showing that $\overline{RS} \leq_{neg,pt} \overline{Int1}$, that $\overline{Int1} \leq_{neg,pt} \overline{Int2}$, and that $\overline{Int2} \leq_{neg,pt} \overline{SIS}$. All reasoning about computational indistinguishability and other cryptographic issues is isolated to the middle level, the proof that $\overline{Int1} \leq_{neg,pt} \overline{Int2}$.

To show that $\overline{Int1} \leq_{neg,pt} \overline{Int2}$, we use results from Section 6. The proofs that $\overline{RS} \leq_{neg,pt} \overline{Int1}$ and that $\overline{Int2}$ implements $\overline{SIS}$ do not involve cryptographic issues. They are reasonably straightforward, using simulation relations of the new kind defined in Section 3.8.

10.1 Simulator structure

For each k, we define a structured simulator $SSim_k$, as the composition of the following five task-PIOAs, with all *send*, *receive*, *rand* and *out''* tasks hidden.

- $TR(D_k, Tdp_k)$, an abstract combination of $Trans(D_k, Tdp_k)$ and $Rec(D_k, Tdp_k, \{Rec\})$.
- $(Src(Tdpp_k)_{tdpp})_k$, isomorphic to $Src(Tdpp_k)$.
- $(Src(\{0,1\} \to D_k)_{yval})_k$, isomorphic to $Src(\{0,1\} \to D_k)$.
- $(Src(\{0,1\})_{bval1})_k$, isomorphic to $Src(\{0,1\})$.

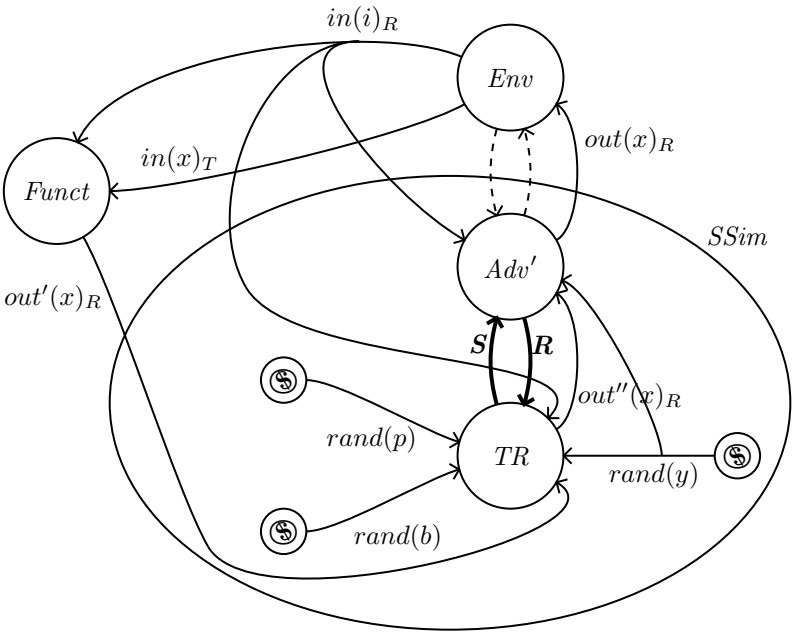

Figure 13: SIS

- Adv'_k, isomorphic to the adversary Adv_k in $(RS)_k$. Adv'_k is identical to Adv except that its $out'(x)_{Rec}$ input actions are renamed to $out''(x)_{Rec}$.

Figure 13 shows $SSim_k$ and its interconnections with the functionality and the environment. (We abbreviated all $send(*)_*$ and $receive(*)_*$ arrows as thick S and R arrows respectively.)

TR has *send* outputs that are inputs to Adv'.

Since *Rec* is corrupted, Adv' sees inputs to *Rec*, and acts as an intermediary for outputs from *Rec*. Thus, Adv' has $in(i)_{Rec}$ inputs, which come from the environment. Adv' has $out''(x)_{Rec}$ inputs, which are outputs of TR, $out(x)_{Rec}$ outputs, which go to the environment, and *receive* outputs, which go to TR. Adv' may also interact with the environment, using other inputs and outputs.

Also, *Funct* provides $out'(x)_{Rec}$ outputs to TR. Thus, TR sees the output produced by *Funct*, which is one of the input bits provided by the environment to *Trans*.

The outputs of Src_{tdpp} and Src_{bval1} go to TR only. The outputs of Src_{yval} go both to TR and to Adv'.

$TR(D, Tdp)$ is defined in Figure 14 and Figure 15. TR plays roles corresponding to those of both *Trans* and *Rec* in the real system. Note that TR produces the *bval* values without using the inverse of the trap-door permutation. It can do this because it knows the receiver's input value and the *yval* values.

We define SIS_k, the structured ideal system, to be the composition $Funct_k \| SSim_k$, with all the $out'(*)$ actions hidden.

Lemma 10.2 *In every reachable state of SIS_k:*

1. *If $TR_k.inval(Trans) \neq \perp$ then $Funct_k.inval(Trans) \neq \perp$, $Funct_k.inval(Rec) \neq \perp$, and $TR_k.inval(Trans) = Funct_k.inval(Trans)(Funct_k.inval(Rec))$.*

10.2 *Int1*

We derive the definition of $Int1_k$ from SIS_k as before. $TR(D_k, Tdp_k)$ is replaced by $TR1(D_k, Tdp_k)$, whose code appears in Figure 16.

$TR(D, Tdp)$:

Signature:

Input:
- $out'(x)_{Rec}$, $x \in \{0,1\}$
- $in(i)_{Rec}$, $i \in \{0,1\}$
- $rand(p)_{tdpp}$, $p \in Tdpp$
- $rand(y)_{yval}$, $y \in (\{0,1\} \to D)$
- $rand(b)_{bval1}$, $b \in \{0,1\}$
- $receive(1,f)_{Rec}$, $f \in Tdp$
- $receive(2,z)_{Trans}$, $z \in (\{0,1\} \to D)$
- $receive(3,b)_{Rec}$, $b \in (\{0,1\} \to \{0,1\})$

Output:
- $send(1,f)_{Trans}$, $f \in Tdp$
- $send(2,z)_{Rec}$, $z \in (\{0,1\} \to D)$
- $send(3,b)_{Trans}$, $b \in (\{0,1\} \to \{0,1\})$
- $out''(x)_{Rec}$, $x \in \{0,1\}$

Internal:
- $fix - zval_{Rec}$
- $fix - bval_{Trans}$

State:
- $inval(Trans), inval(Rec) \in \{0,1,\perp\}$, initially $\perp$
- $tdpp \in Tdp \cup \{\perp\}$, initially $\perp$
- $yval, zval \in (\{0,1\} \to D) \cup \{\perp\}$, initially $\perp$
- $bval1 \in \{0,1,\perp\}$, initially $\perp$
- $bval \in (\{0,1\} \to \{0,1\}) \cup \{\perp\}$, initially $\perp$
- $received_{tdp} \in \{\perp, \top\}$, initially $\perp$
- $received_z \in \{\perp, \top\}$, initially $\perp$
- $received_b \in \{\perp, \top\}$, initially $\perp$

Figure 14: $TR(D, Tdp)$, for the case where $C = \{Rec\}$ (Signature and State).

We define $Int1_k$ to be the same as SIS_k except that $TR(D_k, Tdp_k)$ is replaced by $TR1(D_k, Tdp_k)$, whose code appears in Figure 16. *TR1* differs from *TR* as follows: *TR1* has input actions $in(x)_{Trans}$, by which it receives transmitter input values directly from the environment. Also, *TR1* does not have an input $rand_{bval1}$ nor a $bval1$ state variable; rather, *TR1* calculates *bval* values as follows: For the chosen index i (the one that it received in the $in(i)_{Rec}$ input), *TR1* uses the hard-core predicate applied to the corresponding *yval*, combined with the transmitter input obtained as output from *Funct*; for this, *TR1* does not need to use the inverse of the trap-door permutation. On the other hand, for the non-chosen index, *TR1* uses the hard-core predicate and the inverse of the trap-door permutation, applied to the *zval* value.

Lemma 10.3 *In every reachable state of* $Int1_k$*:*

1. *If* $TR1_k.inval(Trans) \neq \perp$ *then* $Funct_k.inval(Trans) \neq \perp$, $Funct_k.inval(Rec) \neq \perp$, *and* $TR1_k.inval(Trans) = Funct_k.inval(Trans)(Funct_k.inval(Rec))$.
2. *If* $TR1_k.bval \neq \perp$ *then*
   $TR1_k.tdpp \neq \perp$, $TR1_k.zval \neq \perp$, $TR1_k.inval(Trans) \neq \perp$, $TR1_k.inval2(Trans) \neq \perp$, *and* $TR1_k.inval(Rec) \neq \perp$.
3. *If* $TR1_k.zval \neq \perp$ *then* $Funct_k.inval(Rec) \neq \perp$, $TR1_k.inval(Rec) \neq \perp$, *and* $TR1_k.tdpp \neq \perp$.

10.3 *Int2*

$Int2_k$ is the same as SIS_k, except that:

1. It includes a new random source $(Src(\{0,1\})_{cval1})_k$, which is isomorphic to $Src(\{0,1\})$.
2. $TR(D_k, Tdp_k)$ is replaced by $TR2(D_k, Tdp_k)$, where $TRtwo(D, TDp)$ is identical to $TR1(D, Tdp)$ except that:
   (a) *TR2* includes an extra state variable $cval1 \in \{0,1\}$.
   (b) *TR2* has input action $rand(c)_{cval1}$, which sets $cval1 := c$.

Transitions:

$out'(x)_{Rec}$
Effect:
if $inval(Trans) = \bot$ then $inval(Trans) := x$

$in(i)_{Rec}$
Effect:
if $inval(Rec) = \bot$ then $inval(Rec) := i$

$rand(p)_{tdpp}$
Effect:
if $tdpp = \bot$ then $tdpp := p$

$rand(y)_{yval}$
Effect:
if $yval = \bot$ then $yval := y$

$rand(b)_{bval1}$
Effect:
if $bval1 = \bot$ then $bval1 := b$

$receive(1, f)_{Rec}$
Effect:
if $received_{tdp} = \bot$ then $received_{tdp} := \top$

$receive(2, z)_{Trans}$
Effect:
if $received_z = \bot$ then $received_z := \top$

$receive(3, b)_{Rec}$
Effect:
if $yval \neq \bot$ and $received_b = \bot$
then $received_b := \top$

$fix - zval_{Rec}$
Precondition:
$yval, inval(Rec), tdpp, received_{tdp} \neq \bot$
$zval = \bot$
Effect:
$zval(inval(Rec)) := tdpp.funct(yval(inval(Rec)))$
$zval(1 - inval(Rec)) := yval(1 - inval(Rec))$

$fix - bval_{Trans}$
Precondition:
$yval, zval, inval(Trans), inval(Rec), bval1 \neq \bot$
$received_z \neq \bot$
$bval = \bot$
Effect:
$bval(inval(Rec)) :=$
$B(yval(inval(Rec))) \oplus inval(Trans)$
$bval(1 - inval(Rec)) := bval1$

$out''(x)_{Rec}$
Precondition:
$x = inval(Trans) \neq \bot, received_b \neq \bot$
Effect:
none

$send(1, f)_{Trans}$
Precondition:
$tdpp \neq \bot$, $f = tdpp.funct$
Effect:
none

$send(2, z)_{Rec}$
Precondition:
$z = zval \neq \bot$
Effect:
none

$send(3, b)_{Trans}$
Precondition:
$b = bval \neq \bot$
Effect:
none

Tasks: $\{send(1, *)_{Trans}\}$, $\{send(2, *)_{Rec}\}$, $\{send(3, *)_{Trans}\}$, $\{out''(*)_{Rec}\}$, $\{fix - zval_{Rec}\}$, $\{fix - bval_{Trans}\}$.

Figure 15: $TR(D, Tdp)$, for the case where $C = \{Rec\}$ (Transitions and Tasks).

$TR1(D, Tdp)$:

Signature:

Input:
- $in(x)_{Trans}$, $x \in (\{0,1\} \to \{0,1\})$
- $out'(x)_{Rec}$, $x \in \{0,1\}$
- $in(i)_{Rec}$, $i \in \{0,1\}$
- $rand(p)_{tdpp}$, $p \in Tdpp$
- $rand(y)_{yval}$, $y \in (\{0,1\} \to D)$
- $receive(1, f)_{Rec}$, $f \in Tdp$
- $receive(2, z)_{Trans}$, $z \in (\{0,1\} \to D)$
- $receive(3, b)_{Rec}$, $b \in (\{0,1\} \to \{0,1\})$

Output:
- $send(1, f)_{Trans}$, $f \in Tdp$
- $send(2, z)_{Rec}$, $z \in (\{0,1\} \to D)$
- $send(3, b)_{Trans}$, $b \in (\{0,1\} \to \{0,1\})$
- $out''(x)_{Rec}$, $x \in \{0,1\}$

Internal:
- $fix - zval_{Rec}$
- $fix - bval_{Trans}$

State:
- $inval(Trans), inval(Rec) \in \{0, 1, \bot\}$, initially $\bot$
- $inval2(Trans) \in (\{0,1\} \to \{0,1\}) \cup \{\bot\}$, initially $\bot$
- $tdpp \in Tdpp \cup \{\bot\}$, initially $\bot$
- $yval, zval \in (\{0,1\} \to D) \cup \{\bot\}$, initially $\bot$
- $bval \in (\{0,1\} \to \{0,1\}) \cup \{\bot\}$, initially $\bot$
- $received_{tdp} \in \{\bot, \top\}$, initially $\bot$
- $received_z \in \{\bot, \top\}$, initially $\bot$
- $received_b \in \{\bot, \top\}$, initially $\bot$

Transitions:

$in(x)_{Trans}$
Effect:
  if $inval2(Trans) = \bot$
    then $inval2(Trans) := x$

$out'(x)_{Rec}$, $in(i)_{Rec}$, $rand(p)_{tdpp}$, $rand(y)_{yval}$, $receive(1, f)_{Rec}$, $receive(2, z)_{Trans}$, or $receive(3, b)_{Rec}$,
Effect:
  As for $TR(D, Tdp)$.

$fix - bval_{Trans}$
Precondition:
  $tdpp, zval, inval(Trans), inval2(Trans) \neq \bot$
  $inval(Rec), received_z \neq \bot$,
  $bval = \bot$
Effect:
  $bval(inval(Rec)) :=$
    $B(yval(inval(Rec))) \oplus inval(Trans)$
  $bval(1 - inval(Rec)) :=$
    $B(tdpp.inverse(zval(1 - inval(Rec))))$
      $\oplus inval2(Trans)(1 - inval(Rec))$

$fix - zval_{Rec}$, $out''(x)_{Rec}$, $send(1, f)_{Trans}$, $send(2, z)_{Rec}$, or $send(3, b)_{Trans}$
Precondition:
  As for $TR(D, Tdp)$.
Effect:
  As for $TR(D, Tdp)$.

Tasks: $\{send(1, *)_{Trans}\}$, $\{send(2, *)_{Rec}\}$, $\{send(3, *)_{Trans}\}$, $\{out''(*)_{Rec}\}$, $\{fix - zval_{Rec}\}$, $\{fix - bval_{Trans}\}$.

Figure 16: $TR1(D, Tdp)$, for the case where $C = \{Rec\}$.

(c) The line in $fix - bval$ in which $bval(1 - inval(Rec))$ is determined is replaced by:

$$bval(1 - inval(Rec)) := cval1 \oplus inval2(Trans)(1 - inval(Rec)).$$

Thus, instead of calculating the *bval* value for the non-selected index using the hard-core predicate, *TR2* obtains it by applying $\oplus$ to a bit chosen randomly and the actual x input for that index.

The code for $TR2(D, Tdp)$ appears in Figure 17.

10.4 $\overline{RS}$ implements $\overline{Int1}$

We show:

$TR2(D, Tdp)$:

Signature:

Input:
- $in(x)_{Trans}$, $x \in (\{0,1\} \to \{0,1\})$
- $out'(x)_{Rec}$, $x \in \{0,1\}$
- $in(i)_{Rec}$, $i \in \{0,1\}$
- $rand(p)_{tdpp}, p \in Tdpp$
- $rand(y)_{yval}, y \in (\{0,1\} \to D)$
- $rand(c)_{cval1}, c \in \{0,1\}$
- $receive(1,f)_{Rec}$, $f \in Tdp$
- $receive(2,z)_{Trans}$, $z \in (\{0,1\} \to D)$
- $receive(3,b)_{Rec}$, $b \in (\{0,1\} \to \{0,1\})$

Output:
- $send(1,f)_{Trans}$, $f \in Tdp$
- $send(2,z)_{Rec}$, $z \in (\{0,1\} \to D)$
- $send(3,b)_{Trans}$, $b \in (\{0,1\} \to \{0,1\})$
- $out''(x)_{Rec}$, $x \in \{0,1\}$

Internal:
- $fix - zval_{Rec}$
- $fix - bval_{Trans}$

State:
- $inval(Trans), inval(Rec) \in \{0,1,\bot\}$, initially $\bot$
- $inval2(Trans) \in (\{0,1\} \to \{0,1\}) \cup \{\bot\}$, initially $\bot$
- $tdpp \in Tdpp \cup \{\bot\}$, initially $\bot$
- $yval, zval \in (\{0,1\} \to D) \cup \{\bot\}$, initially $\bot$
- $bval \in (\{0,1\} \to \{0,1\}) \cup \{\bot\}$, initially $\bot$
- $cval1 \in \{0,1,\bot\}$, initially $\bot$
- $received_{tdp} \in \{\bot, \top\}$, initially $\bot$
- $received_z \in \{\bot, \top\}$, initially $\bot$
- $received_b \in \{\bot, \top\}$, initially $\bot$

Transitions:

$in(x)_{Trans}$
Effect:
  if $inval2(Trans) = \bot$ then $inval2(Trans) := x$

$out'(x)_{Rec}$, $in(i)_{Rec}$, $rand(p)_{tdpp}$, $rand(y)_{yval}$
$receive(1,f)_{Rec}$, $receive(2,z)_{Trans}$, or $receive(3,b)_{Rec}$,
Effect:
  As for $TR(D, Tdp)$.

$rand(c)_{cval1}$
Effect:
  if $cval1 = \bot$ then $cval1 := c$

$fix - bval_{Trans}$
Precondition:
  $yval, zval, cval1, inval(Trans), inval2(Trans) \neq \bot$
  $inval(Rec), received_z \neq \bot$
  $bval = \bot$
Effect:
  $bval(inval(Rec)) :=$
    $B(yval(inval(Rec))) \oplus inval(Trans)$
  $bval(1 - inval(Rec)) :=$
    $cval1 \oplus inval2(Trans)(1 - inval(Rec))$

$fix - zval_{Rec}$, $out''(x)_{Rec}$, $send(1,f)_{Trans}$,
$send(2,z)_{Rec}$, or $send(3,b)_{Trans}$
Precondition:
  As for $TR(D, Tdp)$.
Effect:
  As for $TR(D, Tdp)$.

Tasks: $\{send(1,*)_{Trans}\}$, $\{send(2,*)_{Rec}\}$, $\{send(3,*)_{Trans}\}$, $\{out''(*)_{Rec}\}$, $\{fix - zval_{Rec}\}$, $\{fix - bval_{Trans}\}$.

Figure 17: $TR2(D, Tdp)$, for the case where $C = \{Rec\}$.

Lemma 10.4 *For every k, $RS_k \leq_0 Int1_k$.*

We prove Lemma 10.4 by choosing an arbitrary environment Env for RS_k and $Int1_k$, and establishing a simulation relation from $RS_k \| Env$ to $Int1_k \| Env$. Then we appeal to Theorem 3.29, the soundness result for simulation relations. The mapping must reconcile the different ways in which $zval$ gets defined in RS and $Int1$. We also show the following lemma, which is what we need to put the pieces of the proof together:

Lemma 10.5 $\overline{RS} \leq_{neg,pt} \overline{Int1}$.

In the rest of this subsection fix Env, an environment for RS_k and $Int1_k$.

10.4.1 State correspondence

Here we define the correspondence R between the states of $RS \| Env$ and $Int1 \| Env$, which we will show to be a simulation relation in Section 10.4.2.

Let ϵ_1 and ϵ_2 be discrete probability measures on finite executions of $RS \| Env$ and $Int1 \| Env$, respectively satisfying the following property:

- **Trace distribution equivalence:** $tdist(\epsilon_1) = tdist(\epsilon_2)$.

Then we say that $(\epsilon_1, \epsilon_2) \in R$ if and only if all of the following hold:

1. For every $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$:
   (a) $u.Funct.inval(Trans) = s.Trans.inval$.
   (b) $u.Funct.inval(Rec) = s.Rec.inval$.
   (c) If $s.Rec.outval \neq \perp$ then $u.TR1.inval(Trans) = s.Rec.outval$.
   (d) $u.TR1.inval2(Trans) = s.Trans.inval$.
   (e) $u.TR1.inval(Rec) = s.Rec.inval$.
   (f) $u.TR1.tdpp = s.Trans.tdpp$.
   (g) $u.TR1.yval = s.Rec.yval$.
   (h) $u.TR1.zval = s.Rec.zval$.
   (i) $u.TR1.bval = s.Trans.bval$.
   (j) $u.Src_{tdpp} = s.Src_{tdpp}$.
   (k) $u.Src_{yval} = s.Src_{yval}$.
   (l) $u.Adv' = s.Adv$.
   (m) $u.Env = s.Env$.
   (n) $u.TR1.received_{tdp} \neq \perp$ iff $s.Rec.tdp \neq \perp$
   (o) $u.TR1.received_{z} \neq \perp$ iff $s.Trans.zval \neq \perp$
   (p) $s.Rec.outval \neq \perp$ iff $u.TR1.received_{b} \neq \perp$

10.4.2 The mapping proof

Lemma 10.6 *The relation R defined in Section 10.4.1 is a simulation relation from $RS \| Env$ to $Int1 \| Env$. Furthermore, for each step of $RS \| Env$, the step correspondence yields at most two steps of $Int1 \| Env$, that is, there is a mapping corrtasks that can be used with R such that, for every ρ, T, $|corrtasks(\rho, T)| \leq 2$.*

The idea of the proof is as follows. All of the tasks in $RS \| Env$ correspond to the same tasks in $Int1 \| Env$, with two exceptions. The first exception is the $\{fix - bval_{Trans}\}$ task, by which *Trans* in the *RS* system determines the value of *bval*, having already received its own input and a round 2 message. This gets mapped to an output task $\{out'(*)_{Rec}\}$ from *Funct* to *TR1* in the *Int1* system, followed by the $\{fix - bval_{Trans}\}$ task of *TR1*. The second exception is the $\{out'(*)_{Rec}\}$ task, by which *Rec* in the *RS* system outputs its result to *Adv*; this gets mapped to the $\{out''(*)_{Rec}\}$ task from *TR1* to Adv' in the *Int1* system.

Proof. We prove that R is a simulation relation from $RS \| Env$ to $Int1 \| Env$ using the mapping $corrtasks : R^*_{RS\|Env} \times R_{RS\|Env} \rightarrow R^*_{Int1\|Env}$, which is defined as follows:

For any $(\rho, T) \in (R^*_{RS\|Env} \times R_{RS\|Env})$:

- If $T \in \{\{choose - rand_{tdpp}\}, \{rand_{tdpp}\}, \{choose - rand_{yval}\}, \{rand_{yval}\}, \{fix - zval_{Rec}\}, \{send(1, *)_{Trans}\}, \{send(2, *)_{Rec}\}, \{send(3, *)_{Trans}\}, \{out(*)_{Rec}\}\}$, then $corrtasks(\rho, T) = T$.

- If T is an output or internal task of Env or Adv, then $corrtasks(\rho, T) = T$.
- If $T = \{fix - bval_{Trans}\}$ then $corrtasks(\rho, T) = \{out'(*)_{Rec}\}\ \{fix - bval_{Trans}\}$.
- If $T = \{out'(*)_{Rec}\}$ then $corrtasks(\rho, T) = \{out''(*)_{Rec}\}$.

We show that R satisfies the two conditions in Lemma 3.31.
Start condition: It is obvious that the Dirac measures on executions consisting of the unique start states s and u of, respectively, $RS \| Env$ and $Int1 \| Env$ are R-related. Property 1 of R holds because the state components of s and u on which R depends are all $\perp$.
Step condition: Suppose $(\epsilon_1, \epsilon_2) \in R$, $\rho_1 \in R^*_{RS\|Env}$, ϵ_1 is consistent with ρ_1, ϵ_2 is consistent with $full(corrtasks)(\rho_1)$ and $T \in R_{RS\|Env}$. Let $\epsilon'_1 = apply(\epsilon_1, T)$ and $\epsilon'_2 = apply(\epsilon_2, corrtasks(\rho_1, T))$.
Claim 1:

1. The state of Env is the same in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Let q_{Env} denote this state of Env.

   This follows from Property 1m.

2. The state of Adv or Adv' is the same in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Let q_{Adv} denote this state of Adv and Adv'.

   This follows from Property 1l.

Claim 2:

1. If T is an output or internal task of Env, then T is either enabled or disabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$ (simultaneously). Furthermore, if T is enabled in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, then:
   (a) There is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   (b) There is a unique transition of Env from q_{Env} with action a; let $tr_{Env} = (q_{Env}, a, \mu_{Env})$ be this transition.
2. If T is an output or internal task of Adv, then T is either enabled or disabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$ (simultaneously). Furthermore, if T is enabled in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, then:
   (a) There is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   (b) There is a unique transition of Adv from q_{Adv} with action a; let $tr_{Adv} = (q_{Adv}, a, \mu_{Adv})$ be this transition.

We establish the step condition by considering cases based on the value of T. In each case we define a probability measure p on an index set I, and for each $j \in I$, two probability measures ϵ'_{1j} and ϵ'_{2j}, on executions of $RS\|Env$ and $Int1\|Env$ respectively. The rest of the proof consists of showing, for each $j \in I$, that $(\epsilon'_{1j}, \epsilon'_{2j}) \in R$, and that $\epsilon'_1 = \sum_{j \in I} p(j)(\epsilon'_{1j})$ and $\epsilon'_2 = \sum_{j \in I} p(j)(\epsilon'_{2j})$.

In each case, the two summations will follow easily from the definition of $apply(,)$ and the definitions of $p(j)$, ϵ'_{1j}, and ϵ'_{2j}, so we will not mention them within the individual cases. More specifically, in each proof case, p satisfies one of the following conditions: (1) p is the Dirac measure on $I = \{1\}$, (2) p is the uniform probability distribution on a finite set I of indices, or (3) p is a probability distribution on a countable set I such that, for every $j \in I$, $p(j) = \mu(x_j)$, where μ is a fixed probability distribution and x_j is an element in $supp(\mu)$ that is defined within the proof case. Whenever (1) holds, ϵ'_1 and ϵ'_2 are defined to be ϵ'_{11} and ϵ'_{21}, respectively, so the summation clearly holds. Whenever (2) holds, the first summation follows from the following facts: (a) Each execution $\alpha \in supp(\epsilon'_1)$ is in $supp(\epsilon'_{1j})$ for a unique j; for every $j' \neq j$, $\epsilon'_{1j'}(\alpha) = 0$. (b) For each execution $\alpha \in supp(\epsilon'_1)$, $\epsilon'_1(\alpha) = p(j)\epsilon'_{1j}(\alpha)$ for the unique j in property (a); this is because $apply(,)$ causes a choice from a uniform distribution and because of the way ϵ'_{1j} is defined. The second summation holds for similar reasons. The reasoning for case (3) is similar to that for case (2), but using μ instead of the uniform distibution.

1. $T = \{choose - rand_{tdpp}\}$.

   We first show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Fix any pair of states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$. Task $T = corrtasks(\rho_1, T)$ is enabled in s (resp. u) iff $s.Src_{tdpp}.chosenval = \perp$ (resp. $u.Src_{tdpp}.chosenval = \perp$). Property 1j implies that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, as needed.

   (a) T is disabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   Let I be the singleton index $\{1\}$, let p be the Dirac measure on 1 and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$. By Definition 3.3 we have $\epsilon'_1 = \epsilon_1$, $\epsilon'_2 = \epsilon_2$. Since $\epsilon_1 \; R \; \epsilon_2$, we have $\epsilon'_{11} \; R \; \epsilon'_{21}$, as needed. The trace distribution equivalence condition $tdist(\epsilon'_1) = tdist(\epsilon'_2)$ also holds since $tdist(\epsilon_1) = tdist(\epsilon_2)$. The summation clearly holds.

   (b) T is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   We define the probability measures needed to show the step correspondence. Let p be the uniform probability measure on the index set $I = \{1 \cdots r\}$ where $r = |Tdp|$. That is, $p(j) = 1/r$ for each $j \in I$. For each $j \in I$, we define probability measure ϵ'_{1j} as follows. The support $supp(\epsilon'_{1j})$ is the set of executions $\alpha \in supp(\epsilon'_1)$ such that $lstate(\alpha).Src_{tdpp}.chosenval$ is the jth element in domain Tdp. For each $\alpha \in supp(\epsilon'_{1j})$ of the form $\alpha' \, choose - rand_{tdpp} \, q$, let $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$. We define ϵ'_{2j} analogously from ϵ'_2.
   Now fix $j \in I$; we show that $(\epsilon'_{1j}, \epsilon'_{2j}) \in R$. To do this, we establish Property 1 of R for ϵ'_{1j} and ϵ'_{2j}, and show trace distribution equivalence for ϵ'_{1j} and ϵ'_{2j}.
   To establish Property 1, consider any states $s' \in supp(lstate(\epsilon'_{1j}))$ and $u' \in supp(lstate(\epsilon'_{2j}))$. By definitions of ϵ'_{1j} and ϵ'_{2j}, we know that $u'.Src_{tdpp}.chosenval = s'.Src_{tdpp}.chosenval$. Hence, Property 1j holds. Since no component other than $Src_{tdpp}.chosenval$ is updated by the application of T, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.
   The fact that $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_{1j} and ϵ'_{2j}.

2. $T = \{rand(*)_{tdpp}\}$.
   We first show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. This part of the proof is identical to the case where $T = choose - random_{tdpp}$.

   (a) T is disabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   This part of the proof is identical to the case where $T = choose - rand_{tdpp}$.

   (b) T is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   We show that there is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. We know by Property 1j that the state of Src_{tdpp} is the same in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Let q denote this state of Src_{tdpp}. By the next-action determinism property for Src_{tdpp} we know that there is a unique action $a \in T$ that is enabled in q. Since T is an output task of Src_{tdpp}, a is also the unique action in T that is enabled in each state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   The probability measures for this case are trivial: Let I be the singleton index set $\{1\}$, let p be the Dirac measure on 1, and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$. To show that $(\epsilon'_1, \epsilon'_2) \in R$, we establish Property 1 of R for ϵ'_1 and ϵ'_2, and show trace distribution equivalence for ϵ'_1 and ϵ'_2.
   To establish Property 1, consider any states $s' \in supp(lstate(\epsilon'_1))$ and $u' \in supp(lstate(\epsilon'_2))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, a, \mu_s) \in D_{RS \| Env}$. Similarly, let u be any state in $supp(lstate(\epsilon_2))$ such that $u' \in supp(\mu_u)$ where $(u, a, \mu_u) \in D_{Int1 \| Env}$.
   By definitions of RS and $Int1$ we know that application of T updates $Trans.tdpp$ in the RS system, and $TR1.tdpp$ in the $Int1$ system. We know by Property 1f that $u.TR1.tdpp =$

$s.Trans.tdpp$. By the effects of T in $Trans$ and $TR1$, we know that $u'.TR1.tdpp = s'.Trans.tdpp$; hence, Property 1f holds. Since no component other than $Trans.tdpp$ in the RS system and $TR1.tdpp$ in the $Int1$ system is updated by the application of T, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.

The fact that $tdist(\epsilon'_1) = tdist(\epsilon'_2)$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_1 and ϵ'_2.

3. $T = \{choose - rand_{yval}\}$.
This case is analogous to the case where $T = \{choose - rand_{tdpp}\}$. In the argument for enabling we use Property 1k instead of Property 1j. In showing the step correspondence, we use the domain $\{0,1\} \to D$ instead of Tdp and also use Property 1k instead of Property 1j.

4. $T = \{rand(*)_{yval}\}$.

   We show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$ using an argument analogous to the one for $T = \{rand(p)_{tdpp}\}$. Here we use Property 1k instead of Property 1j.

   (a) T is disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   This part of the proof is identical to the case where $T = choose - random_{tdpp}$.

   (b) T is enabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   In this case the application of T may cause probabilistic branching in Adv and Adv'. We define the probability measures needed to show the step correspondence. Suppose that $supp(\mu_{Adv})$ is the set $\{q_j : j \in I\}$ of states of Adv, where I is a countable index set. Let p be the probability measure on the index set I such that, for each $j \in I$, $p(j) = \mu_{Adv}(q_j)$. For each $j \in I$, we define probability measure ϵ'_{1j} as follows. The support $supp(\epsilon'_{1j})$ is the set of executions $\alpha \in supp(\epsilon'_1)$ such that $lstate(\alpha).Adv = q_j$. For each $\alpha \in supp(\epsilon'_{1j})$ of the form $\alpha'\ a\ q_j$, let $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$. We define ϵ'_{2j} analogously from ϵ'_2.
   Now fix $j \in I$; it remains to show that $(\epsilon'_{1j}, \epsilon'_{2j}) \in R$. To do this, we establish Property 1 of R for ϵ'_{1j} and ϵ'_{2j}, and show trace distribution equivalence for ϵ'_{1j} and ϵ'_{2j}.
   To establish Property 1, consider any states $s' \in supp(lstate(\epsilon'_{1j}))$ and $u' \in supp(lstate(\epsilon'_{2j}))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, a, \mu_s) \in D_{RS \| Env}$. Similarly, let u be any state in $supp(lstate(\epsilon_2))$ such that $u' \in supp(\mu_u)$ where $(u, a, \mu_u) \in D_{Int1 \| Env}$.
   If $s.Rec.yval \neq \perp$ then by Property 1g, $u.TR1.yval \neq \perp$. In this case, task T has no effect on any component other than Adv, Adv' in either system. Since $s'.Adv = q_j = u'.Adv'$ by definition, it is easy to see that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.
   Now suppose that $s.Rec.yval = \perp$. Then again by Property, 1g $u.TR1.yval = \perp$. Then by the definitions of RS and $Int1$, we know that application of T updates $Rec.yval$ in the RS system, and $TR1.yval$ in the $Int1$ system. It also updates the states of Adv and Adv'.
   We know by Property 1g that $TR1.yval = Rec.yval$ and by 1l that $u.Adv = s.Adv'$. By the effects of T in definitions of Rec and $TR1$, we know that $u'.TR1.yval = s'.Rec.yval$, hence, Property 1g holds for s' and u'. We also know that 1l holds by definition of ϵ'_{1j} and ϵ'_{2j}. Since no component other than $Rec.inval$ and Adv in the RS system, and $TR1.yval$, and Adv' in the $Int1$ system, is updated by the application of T, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.
   The fact that $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_{1j} and ϵ'_{2j}.

5. $T = \{fix - zval_{Rec}\}$.
We first show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Fix any pair of states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$. First, assume that T is enabled

in s. This implies that $s.Rec.yval \neq \perp$, $s.Rec.inval \neq \perp$, $s.Rec.tdp \neq \perp$, $s.Rec.zval = \perp$. Since $s.Rec.yval \neq \perp$, by Property 1g we have $u.TR1.yval \neq \perp$. Since $s.Rec.inval \neq \perp$, by Property 1e, we have $u.TR1.inval(Rec) \neq \perp$. Since $s.Rec.tdp \neq \perp$, by Lemma 5b, we have $s.Rec.tdp = s.Trans.tdpp$ and by Property 1f we have $u.TR1.tdpp \neq \perp$. Since $s.Rec.tdp \neq \perp$, by Property 1n we have $u.TR1.received_{tdp} \neq \perp$. Since $s.Rec.zval = \perp$, by Property 1h we have $u.TR1.zval = \perp$. Hence, T is enabled in u, as needed.

Now assume that T is disabled in s. We need to show that if any of the preconditions of $fix - zval_{Rec}$ is false in s then at least one of the preconditions in u is false. If $s.Rec.yval = \perp$ then by Property 1g, we have $u.TR1.yval = \perp$ If $s.Rec.inval = \perp$, then by Property 1e, we have $u.TR1.inval(Rec) = \perp$. If $s.Rec.tdp = \perp$, then by Property 1n, we have $u.TR1.received_{tdp} = \perp$. If $s.Rec.zval \neq \perp$, by Property 1h we have $u.TR1.zval \neq \perp$. Hence T is disabled in u, as needed.

(a) T is disabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
This part of the proof is identical to the case where $T = choose - rand_{tdpp}$.

(b) T is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
The rest of the proof is easy because *zval* is computed in the same way in both the *RS* and *Int1* systems. The only difference is that in the *Int1* system, the $funct$ component of a trap-door permutation pair is used, whereas in the *RS* system this pair is not available but only a function. The correspondence between the $tdpp.funct$ component of *TR1* and the *tdp* value of *Rec* is established using Lemma 5b.

6. $T = \{fix - bval_{Trans}\}$. In this case, we cannot show uniform enabling or disabling of the corresponding task sequence as in previous cases. Let $\epsilon_2'' = apply(\epsilon_2, out'(x)_{Rec})$. We examine the following two cases:

(a) T is enabled in $supp(lstate(\epsilon_1))$.
We show that the sequence of tasks $\{out'(x)_{Rec}\}\{fix - bval_{Trans}\}$ is enabled in $supp(lstate(\epsilon_2))$. First, consider any state $u \in supp(lstate(\epsilon_2))$; we show that $\{out'(x)_{Rec}\}$ is enabled in u. Choose any $s \in supp(lstate(\epsilon_1))$. Since T is enabled in s and T is an internal task of *Trans*, T is enabled in $s.Trans$. By the precondition of $fix - bval_{Trans}$ in *Trans*, we know that $s.Trans.tdpp \neq \perp$, $s.Trans.zval \neq \perp$, $s.Trans.inval \neq \perp$, and $s.Trans.bval = \perp$. By Properties 1a, 1b, and Lemma 10.3, we have $u.Funct.inval(Trans) \neq \perp$ and $u.Funct.inval(Rec) \neq \perp$. This implies that the action $out'(x)_{Rec}$ is enabled in u, as needed.

We now show that $fix - bval_{Trans}$ is enabled in $supp(lstate(\epsilon_2''))$. So consider any state $u'' \in supp(lstate(\epsilon_2''))$. Choose $u \in supp(lstate(\epsilon_2))$ such that $u'' \in supp(\mu_u)$ where $(u, fix - bval_{Trans}, \mu_u) \in D_{Int1 \| Env}$. Choose any $s \in supp(lstate(\epsilon_1))$. Since $fix - bval_{Trans}$ is enabled in s, we have $s.Trans.tdpp \neq \perp$, $s.Trans.zval \neq \perp$, $s.Trans.inval \neq \perp$, and $s.Trans.bval = \perp$. Then we have $u.TR1.tdpp \neq \perp$, by Property 1f applied to s and u. And $u.TR1.zval \neq \perp$, by Property 1h and Lemma 7b. And $u.TR1.inval2(Trans) \neq \perp$, by Property 1d. And $u.TR1.inval(Rec) \neq \perp$, by Lemma 7b and 6 and Property 1e. And $u.TR1.received_z \neq \perp$, by Property 1o. And finally, $u.TR1.bval = \perp$, by Property 1i. Since the only effect of $out'(x)_{Rec}$ is to set $inval(Trans)$ in *TR1* to x if $inval(Trans) = \perp$, we know that $u''.TR1.inval(Trans) \neq \perp$, and also that $u''.TR1.tdpp \neq \perp$, $u''.TR1.zval \neq \perp$, $u''.TR1.inval2(Trans) \neq \perp$, $u''.TR1.inval(Rec) \neq \perp$, $u''.TR1.inval(Rec) \neq \perp$, $u''.TR1.received_z \neq \perp$, and $u''.TR1.bval = \perp$. Combining all these conditions, we see that $fix - bval_{Trans}$ is enabled in u'', as needed. Next, we define the probability measures. Let I be the singleton index set $\{1\}$, let p be the Dirac measure on 1, and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$. To show that $(\epsilon'_1, \epsilon'_2) \in R$, we establish Property 1 of R for ϵ'_1 and ϵ'_2, and show trace distribution equivalence.

To establish Property 1, consider any states $s' \in supp(lstate(\epsilon'_1))$ and $u' \in supp(lstate(\epsilon'_2))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, fix - bval_{Trans}, \mu_s) \in D_{RS \| Env}$. Let u'' be any state in $supp(lstate(\epsilon_2''))$ such that $u' \in supp(\mu'_u)$ where $(u'', out'(x)_{Rec}, \mu'_u) \in$

$D_{Int1\|Env}$. Let u be any state in $supp(lstate(\epsilon_2))$ such that $u'' \in supp(\mu_u)$ where $(u, out'(x)_{Rec}, \mu_u) \in D_{Int1\|Env}$.

We first show that $s'.Trans.bval = u'.TR1.bval$. By the effect of T, we know that for $i \in \{0,1\}$, $s'.Trans.bval(i) = B(s.Trans.tdpp.inverse(s.Trans.zval(i))) \oplus s.Trans.inval(i)$. All state variables other than *bval* are unchanged in moving from s to s'.

Also, by the effects of the $out'(x)_{Rec}$ and $fix - bval_{Trans}$ actions and by Lemma 10.3 (for the second equality),

$u'.TR1.bval(u.TR1.inval(Rec))$

$= B(u.TR1.yval(u.TR1.inval(Rec))) \oplus u''.TR1.inval(Trans)$

$= B(u.TR1.yval(u.TR1.inval(Rec))) \oplus u''.Funct.inval(Trans)(u''.Funct.inval(Rec))$

$= B(u.TR1.yval(u.TR1.inval(Rec))) \oplus u.Funct.inval(Trans)(u.Funct.inval(Rec))$

Also, we have

$u'.TR1.bval(1 - u.TR1.inval(Rec))$

$= B(u.TR1.tdpp.inverse(u.TR1.zval(1 - u.TR1.inval(Rec))))$

$\oplus\, u.TR1.inval2(Trans)(1 - u.TR1.inval(Rec))$.

In moving from u to u', $TR1.inval(Trans)$ is updated to a non-$\perp$ value and all other state variables except *bval* are unchanged.

To show that $s'.Trans.bval = u'.TR1.bval$, we consider the two indices separately:

i. $i = s.Rec.inval$
   Then by Property 1e, $i = u.TR1.inval(Rec)$. In this case, we must show that $B(s.Trans.tdpp.inverse(s.Trans.zval(i))) \oplus s.Trans.inval(i) = B(u.TR1.yval(u.TR1.inval(Rec))) \oplus u.Funct.inval(Trans)(u.Funct.inval(Rec))$, that is, that $B(s.Trans.tdpp.inverse(s.Trans.zval(i))) \oplus s.Trans.inval(i) = B(u.TR1.yval(i)) \oplus u.Funct.inval(Trans)(u.Funct.inval(Rec))$.
   Now, $s.Trans.inval(i) = s.Trans.inval(s.Rec.inval)$, which is in turn equal to $u.Funct.inval(Trans)(u.Funct.inval(Rec))$. by Properties 1a and 1b for s and u. And $s.Trans.tdpp.inverse(s.Trans.zval(i))) = s.Rec.yval(i)$, by Lemma 10, which is equal to $u.TR1.yval(i))$ by Property 1g. Thus, $s.Trans.tdpp.inverse(s.Trans.zval(i))) = u.TR1.yval(i))$, and so $B(s.Trans.tdpp.inverse(s.Trans.zval(i))) = B(u.TR1.yval(i))$. Combining the equations yielded the needed equation $B(s.Trans.tdpp.inverse(s.Trans.zval(i))) \oplus s.Trans.inval(i) = B(u.TR1.yval(i)) \oplus u.Funct.inval(Trans)(u.Funct.inval(Rec))$.

ii. $i = 1 - s.Rec.inval$
   Then $i = 1 - u.TR1.inval(Rec)$ by Property 1e. In this case, we must show that $B(s.Trans.tdpp.inverse(s.Trans.zval(i))) \oplus s.Trans.inval(i) = B(u.TR1.tdpp.inverse(u.TR1.zval(1 - u.TR1.inval(Rec)))) \oplus u.TR1.inval2(Trans)(1 - u.TR1.inval(Rec))$, that is, that $B(s.Trans.tdpp.inverse(s.Trans.zval(i))) \oplus s.Trans.inval(i) = B(u.TR1.tdpp.inverse(u.TR1.zval(i))) \oplus u.TR1.inval2(Trans)(i)$.
   Now, $s.Trans.inval(i) = u.TR1.inval2(Trans)(i)$ by Property 1d. And $s.Trans.tdpp = u.TR1.tdpp$ by Property 1f. And $s.Trans.zval = u.TR1.zval$ by Property 1h and Lemma 7. It follows that $s.Trans.tdpp.inverse(s.Trans.zval(i)) = u.TR1.tdpp.inverse(u.TR1.zval(i))$, and so $B(s.Trans.tdpp.inverse(s.Trans.zval(i))) = B(u.TR1.tdpp.inverse(u.TR1.zval(i)))$. Combining the equations yields $B(s.Trans.tdpp.inverse(s.Trans.zval(i))) \oplus s.Trans.inval(i) = B(u.TR1.tdpp.inverse(u.TR1.zval(i))) \oplus u.TR1.inval2(Trans)(i)$, as needed.

Thus, we have shown that $s'.Trans.bval = u'.TR1.bval$. To see that Property 1 holds for s' and u', note that it holds for s and u, and the only changes are in the new assigments to *bval* (which are equal, as just shown), and in setting $u'.TR1.inval(Trans)$ to a non-$\perp$ value. The only part of Property 1 that mentions $u'.TR1.inval(Trans)$ is 1c; thus, to see that Property 1 holds for s' and u' (and hence for ϵ_1' and ϵ_2'), it suffices to show that Property 1c holds for s' and u'.

So, suppose that $s'.Rec.outval \neq \perp$. Then $s'.Rec.outval = s.Rec.outval$, which is equal to $s.Trans.inval(s.Rec.inval)$ by Lemma 9b.

This in turn equals $u.Funct.inval(Trans)(u.Funct.inval(Rec))$ by Properties 1a and 1b for s and u, which is equal to $u'.Funct.inval(Trans)(u'.Funct.inval(Rec))$.
Since we know that $u'.TRone.inval(Trans) \neq \perp$, Lemma 10.3 implies that $u'.TRone.inval(Trans) = u'.Funct.inval(Trans)(u'.Funct.inval(Rec))$.
Combining all the equations, we obtain that $s'.Rec.outval = u'.TRone.inval(Trans)$, as needed for 1c.

The fact that $tdist(\epsilon'_1) = tdist(\epsilon'_2)$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_1 and ϵ'_2.

(b) T is disabled in $supp(lstate(\epsilon_1))$.
In this case, the enabling of $out'(x)_{Rec}$ in $supp(lstate(\epsilon_2))$ cannot be deduced by looking at the precondition of $fix - bval_{Trans}$ of $Trans$ and using R. If $Trans.inval = \perp$ and $Rec.inval = \perp$, then by Property 1a and Property 1b, we have $Funct.inval(Trans) = \perp$ and $Funct.inval(Rec) = \perp$, and hence $out'(x)_{Rec}$ is disabled in $supp(lstate(\epsilon_2))$. However if it is not the case that $Trans.inval = \perp$ and $Rec.inval = \perp$ Property 1a and Property 1b are not sufficient to determine whether $out'(x)_{Rec}$ is enabled or disabled in $supp(lstate(\epsilon_2))$.

First suppose that $out'(x)_{Rec}$ is disabled in $supp(lstate(\epsilon_2))$. Let $\epsilon''_2 = apply(\epsilon_2, out'(x)_{Rec})$. We know by Definition 3.3 that $\epsilon''_2 = \epsilon_2$. We can show that $fix - bval_{Trans}$ is disabled in $supp(lstate(\epsilon''_2))$. This follows from Properties 1f,1h,1o,1a,1d,1i, since the negation of any of the preconditions of $fix - bval_{Trans}$ in $Trans$, implies the negation of one of the preconditions of $fix - bval_{Trans}$ in $TR1$. Then we can show Property 1 and trace distibution equivalence condition by letting I be the singleton index $\{1\}$, as in the first case for $choose - random_{tdpp}$.

Now suppose that $out'(x)_{Rec}$ is enabled in $supp(lstate(\epsilon_2))$. Let $\epsilon''_2 = apply(\epsilon_2, out'(x)_{Rec})$. We can show that $fix - bval_{Trans}$ is disabled in $supp(lstate(\epsilon''_2))$. Consider any state $u'' \in supp(lstate(\epsilon''_2))$. Choose $u \in supp(lstate(\epsilon_2))$ such that $u'' \in supp(\mu_u)$ where $(u, out'(x)_{Rec}, \mu_u) \in D_{Int1 \| Env}$. Choose any $s \in supp(lstate(\epsilon_1))$. By applying Properties 1f,1h,1o,1a,1d,1i to s and u, we can show that the negation of any of the preconditions of $fix - bval_{Trans}$ in $Trans$, implies the negation of one of the preconditions of $fix - bval_{Trans}$ in $TR1$. Note that we do not make use of $TR1.inval(Trans)$ in showing this, which is the only variable that may be updated by $out'(x)_{Rec}$ in moving from ϵ_2 to ϵ''_2.

Next, we define the probability measures. Let I be the singleton index set $\{1\}$, let p be the Dirac measure on 1, and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$. The only part of Property 1 that mentions $u'.TR1.inval(Trans)$ is 1c; thus, to see that Property 1 holds for s' and u' (and hence for ϵ'_1 and ϵ'_2), it suffices to show that Property 1c holds for s' and u'.

So, suppose that $s'.Rec.outval \neq \perp$. Then $s'.Rec.outval = s.Rec.outval$, which is equal to $s.Trans.inval(s.Rec.inval)$ by Lemma 9b.
This in turn equals $u.Funct.inval(Trans)(u.Funct.inval(Rec))$ by Properties 1a and 1b for s and u, which is equal to $u'.Funct.inval(Trans)(u'.Funct.inval(Rec))$.
Since we know that $u'.TRone.inval(Trans) \neq \perp$, Lemma 10.3 implies that $u'.TRone.inval(Trans) = u'.Funct.inval(Trans)(u'.Funct.inval(Rec))$.
Combining all the equations, we obtain that $s'.Rec.outval = u'.TRone.inval(Trans)$, as needed for 1c.

The fact that $tdist(\epsilon'_1) = tdist(\epsilon'_2)$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_1 and ϵ'_2.

7. $T = \{send(1, *)_{Trans}\}$.
We first show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. This is analogous to case for $T = choose - rand_{tdpp}$. We use Property 1f to show that $Trans.tdpp = TR1.tdpp$ for all states in $supp(lstate(\epsilon_1))$ and $supp(lstate(\epsilon_2))$.

(a) T is disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
Identical to the corresponding part in the case for $T = choose - rand_{tdpp}$.

(b) T is enabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
First, we show that there is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. We know by Property 1f that variables *Trans.tdpp* and *TR1.tdpp* have the same unique value in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Since the parameter f in $send(1, f)_{Trans}$ is defined to be *Trans.tdpp.funct* we conclude that the action $send(1, Trans.tdpp.funct)$ is the unique action in T that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. We use a as a shorthand for $send(1, Trans.tdpp.funct)$ in the rest of the proof for this case.

Let I be the singleton index set $\{1\}$, let p be the Dirac measure on 1, and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$. To show that $(\epsilon'_1, \epsilon'_2) \in R$, we establish Property 1 of R for ϵ'_1 and ϵ'_2, and show trace distribution equivalence for ϵ'_1 and ϵ'_2. To establish Property 1, consider any state $s' \in supp(lstate(\epsilon'_1))$ and $u' \in supp(lstate(\epsilon'_2))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, a, \mu_s) \in D_{RS \| Env}$. Similarly, let u be any state in $supp(lstate(\epsilon_2))$ such that $u' \in supp(\mu_u)$ where $(u, a, \mu_u) \in D_{Int1 \| Env}$.

By definitions of *RS* and *Int1* we know that application of T updates only *Adv.messages* in the *RS* system and $Adv'.messages$ in the *Int1* system. By Property 1l, $u.Adv' = s.Adv$. It is obvious that $u'.Adv' = s'.Adv$ and that Property 1l holds, since *Adv* and Adv' are the same automaton (except for renaming of the out' actions). Since no component other than *Adv.messages* and $Adv'.messages$ is updated, we conclude that Property 1 holds.

The fact that $tdist(\epsilon'_1) = tdist(\epsilon'_2)$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_1 and ϵ'_2.

8. $T = \{send(2, *)_{Rec}\}$.
We first show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. This is analogous to case for $T = choose - random_{tdpp}$. We use Property 1h to show that $Rec.zval = TR1.zval$ for all states in $supp(lstate(\epsilon_1))$ and $supp(lstate(\epsilon_2))$.

   (a) T is disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   Identical to the corresponding part in the case for $T = choose - random_{tdpp}$.

   (b) T is disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   Next, we show that there is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. We know by Property 1h that variables *Rec.zval* and *TR1.zval* have the same unique value in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, and there is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Note that here a is $send(2, z)_{Rec}$ for a fixed value of z.
   The rest is identical to the proof for $T = \{send(1, f)_{Trans}\}$.

9. $T = \{send(3, *)_{Trans}\}$.
We first show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. This is analogous to case for $T = choose - rand_{tdpp}$. We use Property 1i to show that $Trans.bval = TR1.bval$ for all states in $supp(lstate(\epsilon_1))$ and $supp(lstate(\epsilon_2))$.

   (a) T is disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   Identical to the corresponding part in the case for $T = choose - random_{tdpp}$.

   (b) T is enabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   We show that there is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, arguing as in the case for $T = \{send(1, f)_{Trans}\}$. Here, the unique action is determined by fixing the value of parameter b to the value of variables *Trans.bval* and *TR1.bval*, which is the same in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. The rest of the proof is identical to the proof for $T = \{send(1, f)_{Trans}\}$.

10. $T = \{out'(*)_{Rec}\}$. We first show that the corresponding task $\{out''(x)\}$ is enabled in every state in $supp(lstate(\epsilon_2))$ if and only if $\{out'(x)\}$ is enabled in every state in $supp(lstate(\epsilon_1))$. Fix any state

$u \in supp(lstate(\epsilon_2))$. Note that $\{out''(x)_{Rec}\}$ is an output task of *TR1* in the *Int1* system. Choose any $s \in supp(lstate(\epsilon_1))$. First, suppose T is enabled in s. Since T is an output task of *Rec* in the *RS* system, T is enabled in $s.Rec$ and therefore $s.Rec.outval \neq \perp$. Then by Properties 1c and 1p $u.TR1.inval(Trans) \neq \perp$ and $u.TR1.received_b \neq \perp$. So, $\{out''(x)_{Rec}\}$ is enabled in $u.TR1$, and hence in u, as needed. Now suppose that T is disabled in s. This implies $Rec.outval = \perp$. By Property 1p, we get $u.received_b = \perp$ which is sufficient to show that $out''(x)_{Rec}$ is disabled in u, as needed.

(a) T is disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
Identical to the corresponding part in the case for $T = choose - random_{tdpp}$.

(b) T is enabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
Let I be the singleton index set $\{1\}$, let p be the Dirac measure on 1, and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$. In showing Property 1, we use the fact that applications of T in the *RS* system and $\{out''(x)_{Rec}\}$ in the *Int1* system update only the $outval(Rec)$ state variables in both *Adv* and *Adv*′, which preserves Property 1.
The fact that $tdist(\epsilon'_1) = tdist(\epsilon'_2)$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_1 and ϵ'_2.

11. T is a task of *Env*.
Claim 2 implies that T is uniformly enabled or disabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. If T is disabled, we simply define $I = \{1\}$, $\epsilon'_{11} = \epsilon_1$, $\epsilon'_{21} = \epsilon_2$, and we have that $\epsilon'_{11}\ R\ \epsilon'_{21}$ since $\epsilon_1\ R\ \epsilon_2$. Suppose now that T is enabled in in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Claim 2 now implies that there is a unique action a enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$ and that there is a unique transition of *Env* from q_{Env} with action a; let $tr_{Env} = (q_{Env}, a, \mu_{Env})$ be this transition. We distinguish several cases, according to the possible values for a.

(a) a is an input action of *Trans*. This means that $a = in(x)_{Trans}$ for some fixed x.
We define the probability measures needed to show the step correspondence. Suppose that $supp(\mu_{Env})$ is the set $\{q_j : j \in I\}$ of states of *Env*, where I is a countable index set. Let p be the probability measure on the index set I such that, for each $j \in I$, $p(j) = \mu_{Env}(q_j)$. For each $j \in I$, we define probability measure ϵ'_{1j} as follows. The support $supp(\epsilon'_{1j})$ is the set of executions $\alpha \in supp(\epsilon'_1)$ such that $lstate(\alpha).Env = q_j$. For each $\alpha \in supp(\epsilon'_{1j})$ of the form $\alpha'\ a\ q_j$, let $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$. We define ϵ'_{2j} analogously from ϵ'_2.
Now fix $j \in I$; it remains to show that $(\epsilon'_{1j}, \epsilon'_{2j}) \in R$. To do this, we establish Property 1 of R for ϵ'_{1j} and ϵ'_{2j}, and show trace distribution equivalence for ϵ'_{1j} and ϵ'_{2j}.
To establish Property 1, consider any states $s' \in supp(lstate(\epsilon'_{1j}))$ and $u' \in supp(lstate(\epsilon'_{2j}))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, a, \mu_s) \in D_{RS \| Env}$. Similarly, let u be any state in $supp(lstate(\epsilon_2))$ such that $u' \in supp(\mu_u)$ where $(u, a, \mu_u) \in D_{Int1 \| Env}$.
If $s.Trans.inval \neq \perp$ then by Properties 1a and 1d, $u.Funct.inval(Trans) \neq \perp$ and $u.TR1.inval2(Trans) \neq \perp$. In this case, task T has no effect on any component other than *Env*, in either system. Since $s'.Env = q_j = u'.Env$ by definition, it is easy to see that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.
Now suppose that $s.Trans.inval = \perp$. Then again by Properties, 1a and 1d $u.Funct.inval(Trans) = u.TR1.inval2(Trans) = \perp$. Then by the definitions of *RS* and *Int1*, we know that application of T updates $Trans.inval$ in the *RS* system, and $Funct.inval(Trans)$ and $TR1.inval2(Trans)$ in the *Int1* system. It also updates the state of *Env* in both systems.
We know by Property 1a that $u.Funct.inval(Trans) = s.Trans.inval$, by 1d that $u.TR1.inval2(Trans) = s.Trans.inval$, and by 1m that $u.Env = s.Env$. By the effects of T in definitions of *Trans*, *Funct*, and *TR1*, we know that $u'.Funct.inval(Trans) = s'.Trans.inval$, and $u'.TR1.inval2(Trans) = s'.Trans.inval$; hence, Properties 1a and 1d hold for s' and u'.

We also know that 1m holds by definition of ϵ'_{1j} and ϵ'_{2j}. Since no component other than *Trans.inval* and *Env* in the *RS* system, and *Funct.inval*(*Trans*), *TR1.inval*2(*Trans*), and *Env* in the *Int1* system, is updated by the application of T, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.

The fact that $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_{1j} and ϵ'_{2j}.

(b) a is an input action of *Rec* and *Adv*. This means that $a = in(i)_{Rec}$ for some fixed i.

Here, T is shared between *Env* and *Adv* in both systems. In addition, it is an input to *Rec* in the *RS* system and to *TR1* in the *Int1* system. We must consider the probabilistic branching of *Adv* as well as *Env* in this case.

Recall from Claim 1 that the state of *Adv* or *Adv*$'$ is the same in all sates in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, and we let q_{Adv} denote this state. Claim 2 states that there is a unique transition of *Adv* with action a from q_{Adv}. Let $tr_{Adv} = (q_{Adv}, a, \mu_{Adv})$ be this transition.

Next we define the probability measures needed to show the step correspondence. Suppose that $supp(\mu_{Env} \times \mu_{Adv})$ is the set $\{(q_{j1}, q_{j2}) : j \in I\}$ of pairs of states, where I is a countable index set. Let p be the probability measure on the index set I such that, for each $j \in I$, $p(j) = (\mu_{Env} \times \mu_{Adv})(q_{1j}, q_{2j})$. For each $j \in I$, we define probability measure ϵ'_{1j} as follows. The $supp(\epsilon'_{1j})$ is the set of executions $\alpha \in supp(\epsilon'_1)$ such that $lstate(\alpha).Env = q_{1j}$ and $lstate(\alpha).Adv = q_{2j}$. For each $\alpha \in supp(\epsilon'_{1j})$ of the form $\alpha'\ a\ q$, let $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$. We construct ϵ'_{2j} analogously from ϵ'_2.

The rest of the proof for this case follows the proof for $a = in(x)_{Trans}$. The only difference is that in showing Property 1 for ϵ'_{1j} and ϵ'_{2j}, for a fixed j, we use the fact that application of T affects only *Rec.inval*, *Adv*, and *Env* in the *RS* system, and *Funct.inval*(*Rec*), *TR1.inval*(*Rec*), *Adv*$'$, and *Env* in the *Int1* system, and use Properties 1b, 1e, 1l and 1m, instead of 1a, 1d and 1m.

(c) a is an input of *Adv* but not an input of *Rec*.

Claim 2 implies that there is a unique transition of *Adv* with action a from q_{Adv}. Let $tr_{Adv} = (q_{Adv}, a, \mu_{Adv})$ be this transition.

Suppose that $supp(\mu_{Env} \times \mu_{Adv})$ is the set $\{(q_{j1}, q_{j2}) : j \in I\}$ of pairs of states, where I is a countable index set. Let p be the probability measure on the index set I such that, for each $j \in I$, $p(j) = (\mu_{Env} \times \mu_{Adv})(q_{1j}, q_{2j})$. For each $j \in I$, we define probability measure ϵ'_{1j} as follows. The support $supp(\epsilon'_{1j})$ is the set of executions $\alpha \in supp(\epsilon'_1)$ such that $lstate(\alpha).Env = q_{1j}$ and $lstate(\alpha).Adv = q_{2j}$. For each $\alpha \in supp(\epsilon'_{1j})$ of the form $\alpha'\ a\ q$, let $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$. We construct ϵ'_{2j} analogously from ϵ'_2.

In the rest of the proof we proceed as for $a = \{in(x)_{Trans}\}$. The only difference is that in showing Property 1 for ϵ'_{1j} and ϵ'_{2j}, for a fixed j, we use the fact that application of T affects only the states of *Adv*, *Adv*$'$, and *Env* (by definition of the *RS* and *Int1* systems) and use Properties 1l and 1m.

(d) a is an internal action of *Env* or an output action of *Env* that is not an input of *Trans*, *Rec*, or *Adv*.

To show the step correspondence, we proceed as for $a = in(x)_{Trans}$. The only difference is that in showing Property 1 for ϵ'_{1j} and ϵ'_{2j}, for a fixed j, we use the fact that application of T affects only the state of *Env*, and use Property 1m.

For each index j in the decomposition, the fact that $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_{1j} and ϵ'_{2j}.

12. T is a task of *Adv*.

Claim 2 implies that T is either simultaneously enabled or disabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. If T is disabled, we simply define $I = \{1\}$, $\epsilon'_{11} = \epsilon_1$, $\epsilon'_{21} = \epsilon_2$, and we have that $\epsilon'_{11} R \epsilon'_{21}$ since $\epsilon_1 R \epsilon_2$. Suppose now that T is enabled in in every state in $supp(lstate(\epsilon_1)) \cup$

$supp(lstate(\epsilon_2))$. Claim 2 now implies that there is a unique action a enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$ and that there is a unique transition of Adv from q_{Adv} with action a; let $tr_{Adv} = (q_{Adv}, a, \mu_{Adv})$ be this transition. We distinguish several cases, according to possible values for a.

(a) $T = \{out(*)_{Rec}\}$ for a fixed x.
Since T is an output task of Adv, Claim 2 implies that T is enabled in every state in $supp(lstate(\epsilon_2))$, that there is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, and that there is a unique transition $tr_{Adv} = (q_{Adv}, a, \mu_{Adv})$ of Adv from q_{Adv} with action a. (Here μ_{Adv} is a Dirac distribution.) Also, by next-transition determinism, it follows that there is a unique transition of Env with action a from q_{Env}. Let $tr_{Env} = (q_{Env}, a, \mu_{Env})$ be this transition.
To show the step correspondence, we proceed as for the case where T is a task of Env which consists of the action $a = in(x)_{Trans}$, decomposing the measures generated by the application of T according to the resulting state in Env, and using Property 1m to show that Property 1 holds for each component measure.
For each index j in the decomposition, the fact that $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_{1j} and ϵ'_{2j}.

(b) a is an input action of Env that is different from $out(x)$.
Claim 2 states that the state of Env is the same in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, let q_{Env} be this unique state. Also, there is a unique transition of Env with action a from q_{Env}. Let $tr_{Env} = (q_{Env}, a, \mu_{Env})$ be this transition.
To show the step correspondence, we proceed as in proof case where T is a task of Env and the unique action in this task is an input of Adv. using Properties 1l and 1m.
For each index j in the decomposition, the fact that $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_{1j} and ϵ'_{2j}.

(c) a is an input action of Rec.
This means that $a = receive(1, f)_{Rec}$ for some f, or $a = receive(3, b)_{Rec}$ for some f. Suppose first that $a = receive(1, f)_{Rec}$ for some f. The action a is an output of Adv' and input of $TR1$ in $Int1$.
The rest is similar to the proof for $T = \{send(1, f)_{Trans}\}$. The only difference is that in showing that Property 1 holds, we must show that Property 1n holds. We use the fact that application of T updates only $Rec.tdp$ in RS and $TR1.received_tdp$, and its effect is to set these variables to a non-$\perp$ value if they are $\perp$, and Property 1n can be easily seen to holds in this case.
Suppose now that $a = receive(3, b)_{Rec}$ for some b.
The rest of the proof differs from that for $T = \{receive(1, f)_{Rec}\}$ in that in showing that Property 1 holds, we must show that Properties 1c and 1p are preserved. Thus, consider any state $s' \in supp(lstate(\epsilon'_1))$ and $u' \in supp(lstate(\epsilon'_2))$. Let s be some state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, a, \mu_s) \in D_{RS \| Env}$. Similarly, let u be some state in $supp(lstate(\epsilon_2))$ such that $u' \in supp(\mu_u)$ where $(u, a, \mu_u) \in D_{Int1 \| Env}$.
If $s'.Rec.outval \neq \perp$ from the effect of $receive(3, b)_{Rec}$. Then $s'.Rec.outval = s'.Trans.inval(s'.Rec.inval)$ by Lemma 9b, which is equal to $s.Trans.inval(s.Rec.inval)$.
This in turn equals $u.Funct.inval(Trans)(u.Funct.inval(Rec))$ by Properties 1a and 1b for s and u. Now, $s.Trans.bval \neq \perp$, by Lemma 4, so by Property 1i, $u.TR1.bval \neq \perp$. Therefore, by Lemma 10.3, $u.TRone.inval(Trans) \neq \perp$, and again by Lemma 10.3, $u.TRone.inval(Trans) = u.Funct.inval(Trans)(u.Funct.inval(Rec))$. Combining the equations, we obtain $s'.Rec.outval = u.TR1.inval(Trans)$. Since $u'.TR1.inval(Trans) = u.TR1.inval(Trans)$, we obtain $s'.Rec.outval = u'.TR1.inval(Trans)$ which shows Property 1c.
For s and u we have $s.Rec.outval \neq \perp$ iff $u.TR1.received_b \neq \perp$. Moreover by Property 1g, we have $s.Rec.yval = u.TR1.yval$. From the effects of $receive(3, b)_{Rec}$ actions we know that

Property 1n is preserved.

(d) a is an input action of *Trans*. This means that $a = receive(2, z)_{Trans}$ for some z.
The rest of the proof differs from the case for $T = \{receive(1, f)_{Rec}\}$ only in showing that Property 1 holds; here we have to show that Property 1o holds, which is easy by obesrving that the application of T updates $Trans.zval$ in and $TR1.received_z$ in the same way.

(e) a is either an output action of Adv that is not an input action of *Env*, *Trans*, or *Rec*, or is an internal action of Adv.

To show the step correspondence, we proceed as for $a = in(x)_{Trans}$, in the case where T is an output task of *Env*, but using Adv instead of *Env*. In showing Property 1 for ϵ'_{1j} and ϵ'_{2j}, for a fixed j, we use the fact that application of T affects only the state of Adv (by definition of RS and *Int1*) and use Property 1l.
For each index j in the decomposition, the fact that $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_{1j} and ϵ'_{2j}.

□

Proof. (Of Lemma 10.4:)
By Lemma 10.6, R is a simulation relation from $RS_k \| Env$ to $Int1_k \| Env$. Then Theorem 3.29 implies that $tdists(RS_k \| Env) \subseteq tdists(Int1_k \| Env)$. Since *Env* was chosen arbitrarily, this implies (by definition of $\leq_0$) that $RS_k \leq_0 Int1_k$. □

Proof. (Of Lemma 10.5:)
By Lemma 10.6, R is a simulation relation from $RS_k \| Env$ to $Int1_k \| Env$ for which $|corrtasks(\rho, T)| \leq 2$ for every ρ and T. Since that lemma holds for every k and every *Env*, Theorem 4.31 implies that $\overline{RS} \leq_{neg,pt} \overline{Int1}$. □

10.5 *Int1* implements *Int2*

We show:

Lemma 10.7 *Assume that $\overline{Adv}$ is a polynomial-time-bounded family of adversary automata. Then $\overline{Int1} \leq_{neg,pt} \overline{Int2}$.*

In order to prove this lemma, we consider the following two task-PIOA families, $\overline{SInt1}$ and $\overline{SInt2}$, which are subsystems of the $\overline{Int1}$ and $\overline{Int2}$ families respectively:

- $\overline{SInt1} = hide(\overline{TR1} \| \overline{Src_{tdpp}} \| \overline{Src_{zval}}, \{\{rand(*)_{tdpp}\}, \{rand(*)_{zval}\}\})$,
- $\overline{SInt2} = hide(\overline{TR2} \| \overline{Src_{tdpp}} \| \overline{Src_{zval}} \| \overline{Src_{cval1}}, \{\{rand(*)_{tdpp}\}, \{rand(*)_{zval}\}, \{rand(*)_{cval1}\}\})$.

The only difference between these two systems is in the way the *bval* state variable is computed: a hard-core bit used in *TR1* is replace by a random bit. This is exactly the difference between the $\overline{SH}$ and $\overline{SHR}$ systems we defined in Section 6.3, and we stated that $\overline{SH} \leq_{neg,pt} \overline{SHR}$ (as definition of hard-core predicates).

So, in order to use this relation, we define a new polynomial time-bounded task-PIOA family Ifc' which we compose to $\overline{SH}$ and $\overline{SHR}$ in order to mimic $\overline{SInt1}$ and $\overline{SInt2}$ respectively.

10.5.1 Using hard-core predicate definition

We define new $\overline{SHOT'}$ and $\overline{SHROT'}$ families.

Definition 10.8 *The task-PIOA family $\overline{SHOT'}$ is defined as*

$$hide(\overline{SH} \| \overline{Src_{yval'}} \| \overline{Ifc'}, \{\{rand(*)_{tdp}\}, \{rand(*)_{zval}\}, \{rand(*)_{bval}\}, \{rand(*)_{yval'}\}\}),$$

where

- $\overline{SH}$ *is given in Def. 6.2,*
- $\overline{Src_{yval'}} = \{(Src_{yval'})_k\}_{k \in \mathsf{N}}$*, where each* $(Src_{yval'})_k$ *is isomorphic to* $Src(D_k)$*,*
- $\overline{Ifc'}$ *is defined in Fig. 18 and 19.*

$Ifc'(Tdp, D)$:

Signature:

Input:
- $rand(f)_{tdp}$, $f \in Tdp$
- $rand(z)_{zval}$, $z \in D$
- $rand(y)_{yval'}$, $y \in D$
- $rand(b)_{bval}$, $b \in \{0,1\}$
- $in(x)_{Trans}$, $x \in \{0,1\} \to \{0,1\}$
- $in(i)_{Rec}$, $i \in \{0,1\}$
- $out'(x)_{Rec}$, $x \in \{0,1\}$
- $receive(1, f)_{Rec}$, $f \in Tdp$
- $receive(2, z)_{Trans}$, $z \in (\{0,1\} \to D)$
- $receive(3, b)_{Rec}$, $b \in\in \{0,1\} \to \{0,1\}$

Output:
- $send(1, f)_{Trans}$, $f \in Tdp$
- $send(2, z)_{Rec}$, $z \in \{0,1\} \to D$
- $send(3, b)_{Trans}$, $b \in \{0,1\} \to \{0,1\}$
- $out''(x)_{Rec}$, $x \in \{0,1\}$

Internal:
- $fix - zval_{Rec}$
- $fix - bval_{Trans}$

State:
- $fval \in (Tdp \cup \bot)$, initially $\bot$,
- $zval' \in (D \cup \bot)$, initially $\bot$
- $yval' \in (D \cup \bot)$, initially $\bot$
- $zval \in (\{0,1\} \to D) \cup \{\bot\}$, initially $\bot$
- $bval' \in \{0, 1, \bot\}$, initially $\bot$
- $bval \in (\{0,1\} \to \{0,1\}) \cup \{\bot\}$, initially $\bot$
- $inval(Trans), inval(Rec) \in \{0, 1, \bot\}$, initially $\bot$
- $inval2(Trans) \in (\{0,1\} \to \{0,1\}) \cup \{\bot\}$, initially $\bot$
- $received_{tdp} \in \{\bot, \top\}$, initially $\bot$
- $received_z \in \{\bot, \top\}$, initially $\bot$
- $received_b \in \{\bot, \top\}$, initially $\bot$

Figure 18: Interface, $Ifc'(Tdp, D)$ (Part I)

Definition 10.9 *The task-PIOA family* $\overline{SHROT'}$ *is defined as*

$$hide(\overline{SHR} \| \overline{Src_{yval'}} \| \overline{Ifc'}, \{\{rand(*)_{tdp}\}, \{rand(*)_{zval}\}, \{rand(*)_{bval}\}, \{rand(*)_{yval'}\}\}),$$

where $\overline{SHR}$ *is given in Def. 6.3 while* $\overline{Src_{yval'}}$ *and* $\overline{Ifc'}$ *are as in Def. 10.8.*

Given these two definitions, we have:

Lemma 10.10 $\overline{SHOT'} \leq_{neg,pt} \overline{SHROT'}$.

Proof. By Definition 6.4, $\overline{SH} \leq_{neg,pt} \overline{SHR}$. The task-PIOA families $\overline{Ifc'}$ and $\overline{Src_{yval'}}$ are polynomial-time-bounded. Therefore, since the $\leq_{neg,pt}$ relation is preserved when the related automata are composed with polynomial-time-bounded task-PIOA families (Lemma 4.29),

$$\overline{SH} \| \overline{Ifc'} \| \overline{Src_{yval'}} \leq_{neg,pt} \overline{SHR} \| \overline{Ifc'} \| \overline{Src_{yval'}}.$$

Now, if we define $\overline{U} = \{\{rand(*)_{tdp}\}, \{rand(*)_{zval}\}, \{rand(*)_{bval}\}, \{rand(*)_{yval'}\}\}$, we have that

$$hide(\overline{SH} \| \overline{Ifc'} \| \overline{Src_{yval'}}, \overline{U}) \leq_{neg,pt} hide(\overline{SHR} \| \overline{Ifc'} \| \overline{Src_{yval'}}, \overline{U}),$$

since hiding output tasks of polynomial-time-bounded task-PIOA families preserves the $\leq_{neg,pt}$ relation (Lemma 4.30).

This is equivalent to say that $\overline{SHOT'} \leq_{neg,pt} \overline{SHROT'}$, as needed. □

$Ifc'(Tdp, D)$:

Transitions:

$out'(x)_{Rec}$
Effect:
  if $inval(Trans) = \perp$ then $inval(Trans) := x$

$in(i)_{Rec}$
Effect:
  if $inval(Rec) = \perp$ then $inval(Rec) := i$

$in(x)_{Trans}$
Effect:
  if $inval2(Trans) = \perp$ then $inval2(Trans) := x$

$rand(f)_{tdp}$
Effect:
  if $fval = \perp$ then $fval := f$

$rand(y)_{yval'}$
Effect:
  if $yval' = \perp$ then $yval' := y$

$rand(z)_{zval}$
Effect:
  if $zval' = \perp$ then $zval' := z$

$rand(b)_{bval}$
Effect:
  if $bval' = \perp$ then $bval' := b$

$receive(1, f)_{Rec}$
Effect:
  if $received_{tdp} = \perp$ then $received_{tdp} := \top$

$receive(2, z)_{Trans}$
Effect:
  if $received_z = \perp$ then $received_z := \top$

$receive(3, b)_{Rec}$
Effect:
  if $yval \neq \perp$ and $received_b = \perp$
    then $received_b := \top$

$fix - zval_{Rec}$
Precondition:
  $yval', zval', inval(Rec), fval \neq \perp$
  $received_{tdp} \neq \perp$
  $zval = \perp$
Effect:
  $zval(inval(Rec)) := fval(yval')$
  $zval(1 - inval(Rec)) := zval'$

$fix - bval_{Trans}$
Precondition:
  $bval', yval', zval \neq \perp$
  $inval(Trans), inval2(Trans), inval(Rec) \neq \perp$
  $received_z \neq \perp$
  $bval = \perp$
Effect:
  $bval(inval(Rec)) :=$
    $B(yval') \oplus inval(Trans)$
  $bval(1 - inval(Rec)) :=$
    $bval' \oplus inval2(Trans)(1 - inval(Rec))$

$out''(x)_{Rec}$
Precondition:
  $x = inval(Trans) \neq \perp$, $received_b \neq \perp$
Effect:
  none

$send(1, f)_{Trans}$
Precondition:
  $tdpp \neq \perp$, $f = tdpp.funct$
Effect:
  none

$send(2, z)_{Rec}$
Precondition:
  $z = zval \neq \perp$
Effect:
  none

$send(3, b)_{Trans}$
Precondition:
  $b = bval \neq \perp$
Effect:
  none

Tasks: $\{send(1, *)_{Trans}\}$, $\{send(2, *)_{Rec}\}$, $\{send(3, *)_{Trans}\}$, $\{out''(*)_{Rec}\}$, $\{fix - zval_{Rec}\}$, $\{fix - bval_{Trans}\}$.

Figure 19: Interface, $Ifc'(Tdp, D)$ (Part II)

Some invariants will be helpful in the later proofs:

Lemma 10.11 *In all reachable states of H:*

1. *If $H.zval \neq \perp$ then $H.fval \neq \perp$, $H.yval \neq \perp$ and $H.zval = H.fval(H.yval)$.*
2. *If $H.bval \neq \perp$ then $H.yval \neq \perp$ and $H.bval = B(H.yval)$.*

Lemma 10.12 *In all reachable states of Ifc':*

1. *If $Ifc'.zval \neq \perp$ then $Ifc'.yval' \neq \perp$, $Ifc'.fval \neq \perp$, and $Ifc'.zval' \neq \perp$.*

Lemma 10.13 *In all reachable states of $SHOT'$:*

1. *$Ifc'.fval = H.fval$.*
2. *If $Ifc'.fval \neq \perp$ then $Ifc'.fval = Src_{tdp}.chosenval$.*
3. *If $H.yval \neq \perp$ then $H.yval = Src_{yval}.chosenval$.*
4. *If $Ifc'.yval' \neq \perp$ then $Ifc'.yval' = Src_{yval'}.chosenval$.*
5. *If $Ifc'.zval' \neq \perp$ then $Ifc'.zval' = H.zval$.*
6. *If $Ifc'.bval' \neq \perp$ then $Ifc'.bval' = H.bval$.*

Lemma 10.14 *In all reachable states of $SHROT'$:*

1. *If $Ifc'.fval \neq \perp$ then $Ifc'.fval = Src_{tdp}.chosenval$.*
2. *If $Ifc'.zval' \neq \perp$ then $Ifc'.zval' = Src_{zval}.chosenval$.*
3. *If $Ifc'.yval' \neq \perp$ then $Ifc'.yval' = Src_{yval'}.chosenval$.*
4. *If $Ifc'.bval' \neq \perp$ then $Ifc'.bval' = Src_{bval}.chosenval$.*

10.5.2 The $SInt1$ subsystem implements $SHOT'$

Now, using mappings of the sort we used in Section 10.4.2, we show that $\overline{SInt1} \leq_0 \overline{SHOT'}$ and, in the next subsection, we will prove that $\overline{SHROT'} \leq_0 \overline{SInt2}$. Finally, in Section 10.5.4, we will prove that $\overline{Int1} \leq_{neg,pt} \overline{Int2}$ by using the properties of the mappings we defined and the different properties of the $\leq_{neg,pt}$ relation.

In order to prove $\overline{SInt1} \leq_0 \overline{SHOT'}$, we show that $SInt1_k \leq_0 SHOT'_k$ and $SHROT'_k \leq_0 SInt2_k$ for every k. In the rest of this section, we remove the mention of k everywhere.

Fix any environment Env' for both $SInt1$ and $SHOT'$. We define a simulation relation R from $SInt1 \| Env'$ to $SHOT' \| Env'$.

Let ϵ_1 and ϵ_2 be discrete probability measures on finite executions of $SInt1 \| Env'$ and $SHOT' \| Env'$, respectively, satisfying the trace distribution equivalence and state equivalence properties. Then we say that $(\epsilon_1, \epsilon_2) \in R$ if and only if all of the following hold:

For every $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$:

1. $u.Ifc'.inval(Trans) = s.TR1.inval(Trans)$.
2. $u.Ifc'.inval2(Trans) = s.TR1.inval2(Trans)$.
3. $u.Ifc'.inval(Rec) = s.TR1.inval(Rec)$.
4. if $s.Src_{tdpp}.chosenval = \perp$ then $u.Src_{tdp}.chosenval = \perp$.
5. if $s.Src_{tdpp}.chosenval \neq \perp$ then $u.Src_{tdp}.chosenval = s.Src_{tdpp}.chosenval.funct$.

6. if $s.TR1.tdpp \neq \perp$ then $u.Ifc'.fval = s.TR1.tdpp.funct$ else $u.Ifc'.fval = \perp$.
7. if $s.Src_{yval}.chosenval = \perp$ then $u.Src_{yval}.chosenval = u.Src_{yval'}.chosenval = \perp$
8. if $s.Src_{yval}.chosenval \neq \perp$ then $lstate(\epsilon_2).Src_{yval}.chosenval$ and $lstate(\epsilon_2).Src_{yval'}.chosenval$ are the uniform distribution on D.
9. $s.TR1.yval \neq \perp$ iff $u.H.yval \neq \perp$ and $u.Ifc'.yval' \neq \perp$.
10. if $s.TR1.zval = \perp$ then $u.Ifc'.zval = \perp$ else
    - $u.Ifc'.zval(u.Ifc'.inval(Rec)) = s.TR1.zval(s.TR1.inval(Rec))$ and
    - $u.Ifc'.zval(1 - u.Ifc'.inval(Rec)) = s.TR1.zval(1 - s.TR1.inval(Rec))$.
11. if $s.TR1.bval = \perp$ then $u.Ifc'.bval = \perp$ else
    - $u.Ifc'.bval(u.Ifc'.inval(Rec)) = s.TR1.bval(s.TR1.inval(Rec))$ and
    - $u.Ifc'.bval(1 - u.Ifc'.inval(Rec)) = s.TR1.bval(1 - s.TR1.inval(Rec))$.
12. $u.Ifc'.received_{tdp} = s.TR1.received_{tdp}$.
13. $u.Ifc'.received_z = s.TR1.received_z$.
14. $u.Ifc'.received_b = s.TR1.received_b$.
15. $u.Env' = s.Env'$.

Lemma 10.15 *The relation R defined above is a simulation relation from $SInt1\|Env'$ to $SHOT'\|Env'$. Furthermore, for each step of $SInt1\|Env'$, the step correspondence yields at most three steps of $SHOT'\|Env'$, that is, there is a mapping corrtasks that can be used with R such that, for every ρ, T, $|corrtasks(\rho, T)| \leq 3$.*

Proof. We prove that R is a simulation relation from $SInt1\|Env'$ to $SHOT'\|Env'$ using the mapping $corrtasks(R^*_{SInt1\|Env'} \times R_{SInt1\|Env'}) \rightarrow R^*_{SHOT'\|Env'}$, which is defined as follows:
For any $(\rho, T) \in (R^*_{SInt1\|Env'} \times R_{SInt1\|Env'})$:

- If $T \in \{\{out''(*)_{Rec}\}, \{send(1,*)_{Trans}\}, \{send(2,*)_{Rec}\}, \{send(3,*)_{Trans}\}\}$ then $corrtasks(\rho, T) = T$.
- If T is an output or internal task of Env', then $corrtasks(\rho, T) = T$.
- If $T = \{choose - rand_{tdpp}\}$ then $corrtasks(\rho, T) = \{choose - rand_{tdp}\}$.
- If $T = \{choose - rand_{yval}\}$ then $corrtasks(\rho, T) = \{choose - rand_{yval}\}\{choose - rand_{yval'}\}$.
- If $T = \{rand(*)_{tdpp}\}$ then $corrtasks(\rho, T) = \{rand(*)_{tdp}\}$.
- If $T = \{rand(*)_{yval}\}$ then $corrtasks(\rho, T) = \{rand(*)_{yval}\}\{rand(*)_{yval'}\}$.
- If $T = \{fix - zval_{Rec}\}$ then $corrtasks(\rho, T) = \{fix - zval\}\{rand(*)_{zval}\}\{fix - zval_{Rec}\}$.
- If $T = \{fix - bval_{Trans}\}$ then $corrtasks(\rho, T) = \{fix - bval\}\{rand(*)_{bval}\}\{fix - bval_{Trans}\}$.

We prove that R satisfies the two conditions in Lemma 3.31.
Start condition: It is obvious that the Dirac measures on executions consisting of the unique start states s and u of $SInt1\|Env'$ and $SHOT'\|Env'$, respectively, are R-related: all properties of R holds because the state components of s and u on which R depends are all $\perp$.
Step condition: Suppose $(\epsilon_1, \epsilon_2) \in R$ and T is a task of $SInt1\|Env'$.

Let $\epsilon'_1 = apply(\epsilon_1, T)$ and $\epsilon'_2 = apply(\epsilon_2, corrtasks([lstate(\epsilon_1)], T))$.

The proof follows the same outline as that of Lemma 10.6. Identical versions of Claim 1 and Claim 2 in that proof carry over for Env' to this case. We again consider cases based on the values of T.

1. $T = \{out''(*)_{Rec}\}$.

   Property 1 and Property 14 of R guarantee that T is uniformly enabled or disabled in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.

   If T is uniformly disabled, then this case is similar to the corresponding one in the treatment of the $\{choose - rand_{tdpp}\}$-task in Lemma 10.6.

   Suppose now that T is uniformly enabled in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Property 1 implies that there is only one action a enabled in every state of $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. We distinguish two cases, according to possible environments Env'.

   (a) a is not an input action of Env'.

   The probability measures for this case are trivial: Let I be the singleton index set $\{1\}$, let p be the Dirac measure on 1, and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$. To show that $(\epsilon'_1, \epsilon'_2) \in R$, we observe that executing a does not change any state variable in $SInt1 \| Env'$. The trace distribution equivalence for ϵ'_1 and ϵ'_2 follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_1 and ϵ'_2.

   (b) a is an input action of Env'. Claim 1 states that the state of Env' is the same in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Let $q_{Env'}$ be this state, and $tr_{Env'} = (q_{Env'}, a, \mu_{Env'})$ be the unique transition with action a from $q_{Env'}$.

   Suppose that $supp(\mu'_{Env})$ is the set $\{q_j : j \in I\}$ of states of Env', where I is a countable index set. Let p be the probability measure on the index set I such that, for each $j \in I$, $p(j) = \mu'_{Env}(q_j)$. For each $j \in I$, we define probability measure ϵ'_{1j} as follows. The support $supp(\epsilon'_{1j})$ is the set of executions $\alpha \in supp(\epsilon'_1)$ such that $lstate(\alpha).Env' = q_j$. For each $\alpha \in supp(\epsilon'_{1j})$ of the form $\alpha' \, a \, q_j$, let $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$. We define ϵ'_{2j} analogously from ϵ'_2.

   Now, we see that $(\epsilon'_{1j}, \epsilon'_{2j}) \in R$ because the only modified state variable when applying a are those of Env', our definition of ϵ'_{1j} and ϵ'_{2j} keeps Property 15 true, and the fact that $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_{1j} and ϵ'_{2j}.

2. $T = \{send(1, *)_{Trans}\}$.

   Property 6 of R guarantees that T is uniformly enabled or disabled in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. The rest of this case can be treated in the same way as the previous one.

3. $T = \{send(2, *)_{Rec}\}$.

   Property 10 of R guarantees that T is uniformly enabled or disabled in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. The rest of this case can be treated in the same way as the previous one.

4. $T = \{send(3, *)_{Trans}\}$.

   Property 11 of R guarantees that T is uniformly enabled or disabled in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. The rest of this case can be treated in the same way as the previous one.

5. T is an output or internal task of Env'.

   Claim 2 implies that T is uniformly enabled or disabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. If T is disabled, we simply define $I = \{1\}$, $\epsilon'_{11} = \epsilon_1$, $\epsilon'_{21} = \epsilon_2$, and we have that $\epsilon'_{11} \; R \; \epsilon'_{21}$ since $\epsilon_1 \; R \; \epsilon_2$. Suppose now that T is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Claim 2 now implies that there is a unique action a enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$ and that there is a unique transition of Env from q_{Env} with action a; let $tr_{Env} = (q_{Env}, a, \mu_{Env})$ be this transition. We distinguish several cases, according to the possible values for a.

   (a) $a = in(x)_{Trans}$ for some fixed x.

   This case is similar to the corresponding one in Lemma 10.6: we use the properties 2 and 15 of the definition of R.

(b) $a = in(i)_{Rec}$ for some fixed i.

This case is similar to the corresponding one in Lemma 10.6: we use the properties 3 and 15 of R, and check the properties 10 and 11.

(c) $a = out'(x)_{Rec}$ for some fixed x.

This case is similar to the previous one: we use the properties 1 and 15 of R.

(d) $a = receive(1, f)_{Rec}$ for some fixed f.

This case is similar to the previous one: we use the properties 12 and 15 of R.

(e) $a = receive(2, z)_{Trans}$ for some fixed z.

This case is similar to the previous one: we use the properties 13 and 15 of R.

(f) $a = receive(3, b)_{Rec}$ for some fixed b.

This case is similar to the previous one: we use the properties 9, 14 and 15 of R.

6. $T = \{choose - rand_{tdpp}\}$.

Property 4 and Property 5 of R guarantee that either T is uniformly enabled in all states in $supp(lstate(\epsilon_1))$ and $corrtasks(\rho, T)$ is uniformly enabled in all states in $supp(lstate(\epsilon_2))$, or they are uniformly disabled in all these states.

The rest of this case is similar to the corresponding one in Lemma 10.6, except that we rely on an ordering of the elements of Tdp and $Tdpp$ such that the j-th permutation of Tdp is equal to the $funct$ field of the j-th permutation pair of $Tdpp$.

7. $T = \{choose - rand_{yval}\}$.

Property 7 and Property 8 of R guarantee that either T is uniformly enabled in all states in $supp(lstate(\epsilon_1))$ and the tasks in $corrtasks(\rho, T)$ are enabled in all states in $supp(lstate(\epsilon_2))$, or they are uniformly disabled in all these states.

Property 7 and Property 8 of R are the only properties containing variables modified by the effect of T and $corrtasks(\rho, T)$, and they remain verified after applying those tasks.

8. $T = \{rand(*)_{tdpp}\}$.

Property 4 and Property 5 of R guarantee that either T is uniformly enabled in all states in $supp(lstate(\epsilon_1))$ and $corrtasks(\rho, T)$ is enabled in all states in $supp(lstate(\epsilon_2))$, or they are uniformly disabled in all these states.

Property 6 of R is the only one containing variables modified by the effect of T and $corrtasks(\rho, T)$, and it remains verified after applying those tasks.

9. $T = \{rand(*)_{yval}\}$.

Property 7 and Property 8 of R guarantee that either T is uniformly enabled in all states in $supp(lstate(\epsilon_1))$ and the tasks in $corrtasks(\rho, T)$ are enabled in all states in $supp(lstate(\epsilon_2))$, or they are uniformly disabled in all these states.

Property 9 of R is the only one containing variables modified by the effect of T and $corrtasks(\rho, T)$, and it remains verified after applying those tasks.

10. $T = \{fix - zval_{Rec}\}$.

Properties 9, 3, 6, 12, and 10 guarantee that either T is uniformly enabled or disabled in all states in $supp(lstate(\epsilon_1))$.

Suppose first that T is uniformly enabled in those states. Let u be any state in $supp(lstate(\epsilon_2))$.

(a) Property 9 implies that $u.H.yval \neq \perp$ and $u.Ifc'.yval' \neq \perp$.

(b) Property 3 implies that $u.Ifc'.inval(Rec) \neq \perp$.

(c) Property 6 and Lemma 10.13 imply that $u.Ifc'.fval = u.H.fval \neq \perp$.

(d) Property 12 implies that $u.Ifc'.received_{tdp} \neq \perp$.

(e) Property 10 implies that $u.Ifc'.zval = \perp$.

(f) Items (a) and (c) imply that the $\{fix - zval\}$-task is enabled. After executing this task, we have that $u.H.zval \neq \perp$.

(g) Item (f) implies that the $\{rand(z)_{zval}\}$-task is enabled. After executing this task, we have that $u.Ifc'.zval' \neq \perp$.

(h) Items (a), (g), (b), (c), (d) and (e) imply that the $\{fix - zval_{Rec}\}$-task is enabled.

So, all tasks in the sequence $corrtasks(\rho, T)$ are uniformly enabled. We can verify that $\epsilon_1' R \epsilon_2'$ by using the same properties.

Suppose now that T is uniformly disabled in all states in $supp(lstate(\epsilon_1))$. Let s be any state in $supp(lstate(\epsilon_1))$ and u be any state in $supp(lstate(\epsilon_2))$. We distinguish several cases, according to the variables responsible of disabling T.

- $s.TR1.yval = \perp$. Then, by Property 9, $u.H.yval = \perp$ and, by Lemma 10.13, $u.H.zval = \perp$ and $u.H.bval = \perp$. Therefore, the $\{fix-zval\}$, $\{rand(z)_{zval}\}$, $\{fix-bval\}$ and $\{rand(b)_{bval}\}$ tasks are disabled. Furthermore, by Property 9, $u.Ifc'.yval' = \perp$, which implies that the $\{fix - zval_{Rec}\}$-task is disabled too. So, nothing happens when we apply T and $corrtasks(\rho, T)$ in that case.
- $s.TR1.inval(Rec) = \perp$. In this case:
  (a) The $\{fix-zval\}$-task is either uniformly enabled or disabled. If it is uniformly enabled, the only effect of this task is to change the $u.H.zval$ variable, which preserves R since this variable does not appear in its definition. If it is uniformly disabled, then nothing happens and R is preserved too.
  (b) The $\{rand(z)_{zval}\}$-task is either uniformly enabled or disabled. If it is uniformly enabled, the only effect of this task is to change the $u.Ifc'.zval'$ variable, which preserves R since this variable does not appear in its definition. If it is uniformly disabled, then nothing happens and R is preserved too.
  (c) The $\{fix - zval_{Rec}\}$-task is always disabled, by Property 3.
- $s.TR1.tdpp = \perp$. Property 6 and Lemma 10.13 imply that $u.Ifc'.fval = \perp$, $u.H.fval = \perp$. Therefore, the $\{fix - zval\}$, $\{rand(z)_{zval}\}$, and $\{fix - zval_{Rec}\}$ tasks are disabled.
- $s.TR1.received_{tdp} = \perp$. This case can be treated in the same way as the $inval(Rec) = \perp$ case, if we note that Property 12 implies that $u.Ifc'.received_{tdp} = \perp$.
- $s.TR1.zval \neq \perp$. Property 10 and Lemma 10.13, we know that $u.Ifc'.zval \neq \perp$, $u.Ifc'.fval \neq \perp$, $u.Ifc'.zval' \neq \perp$, $u.H.fval \neq \perp$, $u.H.yval \neq \perp$. This implies that the $\{fix - zval\}$ and $\{rand(z)_{zval}\}$ tasks have no effect, and that the $\{fix - zval_{Rec}\}$ task is disabled.

11. $T = \{fix - bval_{Trans}\}$.

Properties 6, 10, 1, 2, 3, 13, and 11 guarantee that either T is uniformly enabled or disabled in all states in $supp(lstate(\epsilon_1))$.

Suppose first that T is uniformly enabled in those states. Let u be any state in $supp(lstate(\epsilon_2))$.

(a) Property 10 and Lemma 10.13 imply that $u.Ifc'.zval \neq \perp$, $u.Ifc'.yval' \neq \perp$, and $u.Ifc'.bval' \neq \perp$.

(b) Properties 1, 2 and 3 imply that $u.Ifc'.inval(Trans), u.Ifc'.inval2(Trans), u.Ifc'.inval(Rec) \neq \perp$.

(c) Property 13 implies that $u.Ifc'.received_z \neq \perp$.

(d) Property 11 implies that $u.Ifc'.bval = \perp$.

(e) Items (a) implies that the $\{fix - bval\}$-task is enabled. After executing this task, we have that $u.H.bval \neq \perp$.

(f) Item (e) implies that the $\{rand(b)_{bval}\}$-task is enabled. After executing this task, we have that $u.Ifc'.bval' \neq \perp$.

(g) Items (f), (a), (b), (c) and (d) imply that the $\{fix - bval_{Rec}\}$-task is enabled.

So, all tasks in the sequence $corrtasks(\rho, T)$ are uniformly enabled. We can verify that $\epsilon_1' R \epsilon_2'$ by using the same properties.

Suppose now that T is uniformly disabled in all states in $supp(lstate(\epsilon_1))$. Let s be any state in $supp(lstate(\epsilon_1))$ and u be any state in $supp(lstate(\epsilon_2))$. We distinguish several cases, according to the variables responsible of disabling T.

(a) $s.TR1.tdpp = \perp$. Lemma 10.3 implies that $s.TR1.zval = \perp$ too. We refer to that case.

(b) $s.TR1.zval = \perp$. Property 10 implies that $u.Ifc'.zval = \perp$.

i. The $\{fix - bval\}$-task is either uniformly enabled or disabled (by Property 9). If it is uniformly enabled, the only effect of this task is to change the $u.H.bval$ variable, which preserves R since this variable does not appear in its definition. If it is uniformly disabled, then nothing happens and R is preserved too.

ii. The $\{rand(b)_{zval}\}$-task is either uniformly enabled or disabled. If it is uniformly enabled, the only effect of this task is to change the $u.Ifc'.bval'$ variable, which preserves R since this variable does not appear in its definition. If it is uniformly disabled, then nothing happens and R is preserved too.

iii. The $\{fix - bval_{Trans}\}$ task is uniformly disabled.

(c) $s.TR1.inval(Trans) = \perp$. Idem, but using Property 1.

(d) $s.TR1.inval2(Trans) = \perp$. Idem, but using Property 2.

(e) $s.TR1.inval(Rec) = \perp$. Idem, but using Property 3.

(f) $s.TR1.received_z = \perp$. Idem, but using Property 13.

(g) $s.TR1.bval \neq \perp$. Idem, but using Property 11.

□

10.5.3 $SHROT'$ implements the $SInt2$ subsystem

Fix any environment Env' for both $SHROT'$ and $SInt2$. We define a simulation relation R from $SHROT'\|Env'$ to $SInt2\|Env'$.

Let ϵ_1 and ϵ_2 be discrete probability measures on finite executions of $SHROT'\|Env'$ and $SInt2\|Env'$, respectively, satisfying the trace distribution equivalence and state equivalence properties. Then we say that $(\epsilon_1, \epsilon_2) \in R$ if and only if all of the following hold:

For every $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$:

1. $u.TR2.inval(Trans) = s.Ifc'.inval(Trans)$.
2. $u.TR2.inval2(Trans) = s.Ifc'.inval2(Trans)$.
3. $u.TR2.inval(Rec) = s.Ifc'.inval(Rec)$.
4. if $s.Src_{tdp}.chosenval = \perp$ then $u.Src_{tdpp}.chosenval = \perp$.
5. if $s.Src_{tdp}.chosenval \neq \perp$ then $u.Src_{tdpp}.chosenval.funct = s.Src_{tdp}.chosenval$.
6. if $s.Ifc'.fval \neq \perp$ then $u.TR2.tdpp.funct = s.Ifc'.fval$ else $u.TR2.tdpp = \perp$.

7. if $s.Src_{zval}.chosenval = \perp$ or $s.Src_{yval'}.chosenval = \perp$ then $u.Src_{yval}.chosenval = \perp$.
8. if $s.Src_{zval}.chosenval \neq \perp$ and $s.Src_{yval'}.chosenval \neq \perp$ then $lstate(\epsilon_2).Src_{yval}.chosenval$ is the uniform distribution on $(\{0,1\} \to D)$.
9. if $s.Ifc'.zval' = \perp$ or $s.Ifc'.yval' = \perp$ then $u.TR2.yval = \perp$ else $u.TR2.yval \neq \perp$.
10. $s.Src_{bval}.chosenval = u.Src_{cval1}.chosenval$.
11. if $s.Ifc'.bval' = u.TR2.cval1$.
12. if $s.Ifc'.zval = \perp$ then $u.TR2.zval = \perp$ else
    - $u.TR2.zval(u.TR2.inval(Rec)) = s.Ifc'.zval(s.Ifc'.inval(Rec))$ and
    - $u.TR2.zval(1 - u.TR2.inval(Rec)) = s.Ifc'.zval(1 - s.Ifc'.inval(Rec))$.
13. if $s.Ifc'.bval = \perp$ then $u.TR2.bval = \perp$ else
    - $u.TR2.bval(u.TR2.inval(Rec)) = s.Ifc'.bval(s.Ifc'.inval(Rec))$ and
    - $u.TR2.bval(1 - u.TR2.inval(Rec)) = s.Ifc'.bval(1 - s.Ifc'.inval(Rec))$.
14. $u.Ifc'.received_{tdp} = s.TR1.received_{tdp}$.
15. $u.Ifc'.received_{z} = s.TR1.received_{z}$.
16. $u.Ifc'.received_{b} = s.TR1.received_{b}$.
17. $u.Env' = s.Env'$.

Lemma 10.16 *The relation R defined above is a simulation relation from $SHROT'\|Env'$ to $SInt2\|Env'$. Furthermore, for each step of $SHROT'\|Env'$, the step correspondence yields at most one step of $SInt2\|Env'$, that is, there is a mapping corrtasks that can be used with R such that, for every ρ, T, $|corrtasks(\rho,T)| \leq 1$.*

Proof. We prove that R is a simulation relation from $SHROT'\|Env'$ to $SInt2\|Env'$ using the mapping $corrtasks(R^*_{SHROT'\|Env'} \times R_{SHROT'\|Env'}) \to R^*_{SInt2\|Env'}$, which we define as follows:
For any $(\rho, T) \in (R^*_{SHROT'\|Env'} \times R_{SHROT'\|Env'})$:

- If $T \in \{\{out''(*)_{Rec}\}, \{fix{-}zval_{Rec}\}, \{fix{-}bval_{Trans}\}, \{send(1,*)_{Trans}\}, \{send(2,*)_{Rec}\}, \{send(3,*)_{Trans}\}\}$ then $corrtasks(\rho,T) = \{T\}$.
- If T is an output or internal task of Env', then $corrtasks(\rho,T) = T$.
- If $T = \{choose - rand_{tdp}\}$ then $corrtasks(\rho,T) = \{choose - rand_{tdpp}\}$.
- If $T = \{rand(*)_{tdp}\}$ then $corrtasks(\rho,T) = \{rand(*)_{tdpp}\}$.
- If $T = \{choose - rand_{yval'}\}$ then
  - if ρ does not contain $\{choose - rand_{zval}\}$ then $corrtasks(\rho,T) = \lambda$,
  - else $corrtasks(\rho,T) = \{choose - rand_{yval}\}$.
- If $T = \{rand(*)_{yval'}\}$ then
  - if $\{choose - rand_{zval}\}, \{rand(*)_{zval}\}$ is a subsequence of ρ then $corrtasks(\rho,T) = \{choose - rand_{yval}\}$,
  - else $corrtasks(\rho,T) = \lambda$.
- If $T = \{choose - rand_{zval}\}$ then

- if ρ does not contain $\{choose - rand_{yval'}\}$ then $corrtasks(\rho, T) = \lambda$,
- else $corrtasks(\rho, T) = \{choose - rand_{yval}\}$.

- If $T = \{rand(*)_{zval}\}$ then
  - if $\{choose - rand_{yval'}\}, \{rand(*)_{yval}\}$ is a subsequence of ρ then $corrtasks(\rho, T) = \{choose - rand_{yval}\}$,
  - else $corrtasks(\rho, T) = \lambda$.
- If $T = \{choose - rand_{bval}\}$ then $corrtasks(S, T) = \{choose - rand_{cval1}\}$.
- If $T = \{rand(*)_{bval}\}$ then $corrtasks(\rho, T) = \{rand(*)_{cval1}\}$.

We show that R satisfies the two conditions in Lemma 3.31.
Start condition: It is obvious that the Dirac measures on executions consisting of the unique start states s and u of $SHROT'\|Env'$ and $SInt2\|Env'$, respectively, are R-related. All properties of R hold because the state components of s and u on which R depends are all $\perp$.
Step condition: Suppose $(\epsilon_1, \epsilon_2) \in R$ and ρ be a task scheduler for $SInt2\|Env'$, T is a task of $SInt2\|Env'$, and let s be any state in $supp(lstate(\epsilon_1))$.

Let $\epsilon_1' = apply(\epsilon_1, T)$ and $\epsilon_2' = apply(\epsilon_2, corrtasks(\rho, T))$.

The proof follows the same outline as that of Lemma 10.6 and 10.15. Identical versions of Claim 1 and Claim 2 in that proof carry over for Env' to this case. We again consider cases based on the values of T.

1. $T = \{out''(*)_{Rec}\}$.

   We proceed as for the same task in Lemma 10.15.

2. $T = \{fix - zval_{Rec}\}$.

   Properties 9, 3, 6, 14 and 12 guarantee that either T is uniformly enabled or disabled in all states in $supp(lstate(\epsilon_1))$.

   Suppose first that T is uniformly enabled in those states. Let u be any state in $supp(lstate(\epsilon_2))$.

   (a) Property 9 implies that $u.TR2.yval \neq \perp$.
   (b) Property 3 implies that $u.TR2.inval(Rec) \neq \perp$.
   (c) Property 6 implies that $u.TR2.tdpp \neq \perp$.
   (d) Property 14 implies that $u.TR2.received_{tdp} \neq \perp$.
   (e) Property 12 implies that $u.TR2.zval = \perp$.

   So, $corrtasks(\rho, T)$ is uniformly enabled. We can verify that $\epsilon_1' R \epsilon_2'$ by using the same properties.

   The same five properties can be used to check that $corrtasks(\rho, T)$ is uniformly disabled when T is uniformly disabled.

3. $T = \{fix - bval_{Trans}\}$

   Properties 11, 9, 12, 1, 2, 3, 15 and 13 guarantee that either T is uniformly enabled or disabled in all states in $supp(lstate(\epsilon_1))$.

   Suppose first that T is uniformly enabled in those states. Let u be any state in $supp(lstate(\epsilon_2))$.

   (a) Property 11 implies that $u.TR2.cval1 \neq \perp$.
   (b) Property 9 and Lemma 10.12 imply that $u.TR2.yval \neq \perp$.
   (c) Property 12 implies that $u.TR2.zval = \perp$.
   (d) Properties 1, 2 and 3 imply that $u.TR2.inval(Trans), u.TR2.inval2(Trans), u.TR2.inval(Rec) \neq \perp$

(e) Property 15 implies that $u.TR2.received_z \neq \perp$.

(f) Property 13 implies that $u.TR2.bval = \perp$.

So, $corrtasks(\rho, T)$ is uniformly enabled. We can verify that $\epsilon_1' R \epsilon_2'$ by using the same properties.

The same eight properties can be used to check that $corrtasks(\rho, T)$ is uniformly disabled when T is uniformly disabled.

4. $T = \{send(1, *)_{Trans}\}$ We proceed as for the same task in Lemma 10.15.
5. $T = \{send(2, *)_{Rec}\}$ We proceed as for the same task in Lemma 10.15.
6. $T = \{send(3, *)_{Trans}\}$ We proceed as for the same task in Lemma 10.15.
7. $T = \{choose - rand_{tdp}\}$ We proceed as for the $\{choose - rand_{tdpp}\}$ task in Lemma 10.15.
8. $T = \{rand(*)_{tdp}\}$ We proceed as for the $\{rand(*)_{tdpp}\}$ task in Lemma 10.15.
9. $T = \{choose - rand_{yval'}\}$ and $\{choose - rand_{zval}\} \notin \rho$

   The condition on ρ implies that $s.Src_{zval}.chosenval = \perp$. Property 7 implies that T is uniformly enabled or disabled in all states in $supp(lstate(\epsilon_1))$.

   The condition on ρ implies that Property 7 and Property 8 remain verified after applying T.
10. $T = \{choose - rand_{yval'}\}$ and $\{choose - rand_{zval}\} \in \rho$

    The condition on ρ implies that $s.Src_{zval}.chosenval \neq \perp$. Property 7 and Property 8 imply that T is uniformly enabled or disabled in all states in $supp(lstate(\epsilon_1))$.

    Suppose first that T is uniformly enabled. Then, Property 7 implies that $corrtasks(\rho, T)$ is uniformly enabled too. Applying T makes the condition in this property become false, and makes the one in Property 8 become true. The effect of $corrtasks(\rho, T)$ guarantees that Property 8 is verified after applying T and $corrtasks(\rho, T)$.

    Suppose next that T is uniformly disabled. Then, Property 8 implies that $corrtasks(\rho, T)$ is uniformly disabled too.
11. $T = \{rand(*)_{yval'}\}$ and $\{choose - rand_{zval}\}, \{rand(*)_{zval}\}$ is a subsequence of ρ

    The condition on ρ implies that $s.Ifc'zval' \neq \perp$. Property 9 implies that T is uniformly enabled or disabled in all states in $supp(lstate(\epsilon_1))$.

    Suppose first that T is uniformly enabled. Then, Property 7 and the condition on ρ imply that $corrtasks(\rho, T)$ is uniformly enabled too. The effect of $corrtasks(\rho, T)$ guarantees that Property 9 is verified after applying T and $corrtasks(\rho, T)$.

    Suppose next that T is uniformly disabled. Then, Property 7 implies that $corrtasks(\rho, T)$ is uniformly disabled too.
12. $T = \{rand(*)_{yval'}\}$ and $\{choose - rand_{zval}\}, \{rand(*)_{zval}\}$ is not a subsequence of ρ

    The condition on ρ implies that $s.Ifc'zval' = \perp$. Property 9 implies that T is uniformly enabled or disabled in all states in $supp(lstate(\epsilon_1))$.

    Suppose first that T is uniformly enabled. Then, Property 9 remains verified after applying T.
13. $T = \{choose - rand_{zval}\}$. This case is similar to the one of the $T = \{choose - rand_{yval'}\}$-task.
14. $T = \{rand(*)_{zval}\}$. This case is similar to the one of the $T = \{rand(*)_{yval'}\}$-task.
15. $T = \{choose - rand_{bval}\}$. This case is similar to the one of the $T = \{choose - rand_{tdp}\}$-task, but using Property 10.
16. $T = \{rand(*)_{bval}\}$. This case is similar to the one of the $T = \{rand(*)_{tdp}\}$-task, but using Property 11.

□

10.5.4 *Int1* implements *Int2*

Proof. (of Lemma 10.7)
In Lemma 10.15 and 10.16, we proved that $\overline{SInt1} \leq_0 \overline{SHOT'}$ and $\overline{SHROT'} \leq_0 \overline{SInt2}$. Furthermore, the *corrtasks* mappings we used in these proofs only increase the length of the schedules by a constant factor. So, we can use the soundness result of our simulation relation given in Thm. 4.31 to deduce that $\overline{SInt1} \leq_{neg,pt} \overline{SHOT'}$ and $\overline{SHROT'} \leq_{neg,pt} \overline{SInt2}$

Now, since $\overline{SHOT'} \leq_{neg,pt} \overline{SHROT'}$ (see Lemma 10.10) and since the $\leq_{neg,pt}$ implementation relation is transitive (see Lemma 4.28), we obtain $\overline{SInt1} \leq_{neg,pt} \overline{SInt2}$.

Now, by composing $\overline{SInt1}$ and $\overline{SInt2}$ with the polynomial-time bounded task-PIOA families $\overline{Adv}$ and $\overline{Funct}$, and using Lemma 4.29, we obtain:

$$\overline{Funct} \| \overline{Adv} \| \overline{SInt1} \leq_{neg,pt} \overline{Funct} \| \overline{Adv} \| \overline{SInt2}.$$

Now, coming back to the definitions of $\overline{SInt1}$ and $\overline{SInt1}$, we observe that this is equivalent to saying that:

$$hide(\overline{Funct} \| \overline{Adv} \| \overline{TR1} \| \overline{Src_{tdpp}} \| \overline{Src_{zval}}, \{\{rand(*)_{tdpp}\}, \{rand(*)_{zval}\}\})$$
$$\leq_{neg,pt} hide(\overline{Funct} \| \overline{Adv} \| \overline{TR2} \| \overline{Src_{tdpp}} \| \overline{Src_{zval}} \| \overline{Src_{cval1}}, \{\{rand(*)_{tdpp}\}, \{rand(*)_{zval}\}, \{rand(*)_{cval1}\}\})$$

or, in other words, $\overline{Int1} \leq_{neg,pt} \overline{Int2}$, as needed. □

10.6 *Int2* implements *SIS*

We show:

Lemma 10.17 *For every k, $Int2_k \leq_0 SIS_k$.*

We prove Lemma 10.17 by choosing an arbitrary environment *Env* for $Int2_k$ and SIS_k, establishing a simulation relation from $Int2_k \| Env$ to $SIS_k \| Env$, and appealing to Theorem 3.29, the soundness result for simulation relations.

The only differences between *Int2* and *SIS* are that *Int2* uses *TR2* and Src_{cval1} whereas *SIS* uses *TR* and Src_{bval1}. The key difference here is that *TR2* calculates the *bval* value for the non-selected index as the $\oplus$ of a random *cval*1 bit and the real input bit, whereas *TR* chooses it randomly (using *bval*1). Either way, it's a random bit.

We also show:

Lemma 10.18 $\overline{Int2} \leq_{neg,pt} \overline{SIS}$.

10.6.1 State correspondence

Here we define the correspondence R from the states of $Int2 \| Env$ to states of $SIS \| Env$, which we will show to be a simulation relation in Section 10.6.2.

Let ϵ_1 and ϵ_2 be discrete probability measures on finite executions of *Int2* and *SIS*, respectively, satisfying the following property:

1. **Trace distribution equivalence:** $tdist(\epsilon_1) = tdist(\epsilon_2)$.

Then we say that $(\epsilon_1, \epsilon_2) \in R$ if and only if all of the following hold:

1. For every $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$:
   (a) $u.Funct = s.Funct$.
   (b) $u.Funct.inval(Trans) = s.TR2.inval2(Trans)$.
   (c) $u.TR.inval(Trans) = s.TR2.inval(Trans)$.

(d) $u.TR.inval(Rec) = s.TR2.inval(Rec)$.

(e) $u.TR.tdpp = s.TR2.tdpp$.

(f) $u.TR.yval = s.TR2.yval$.

(g) $u.TR.zval = s.TR2.zval$.

(h) If $u.TR.bval \neq \perp$ then $s.TR2.cval1 \neq \perp$, $s.TR2.inval(Trans) \neq \perp$, $s.TR2.inval(Rec)) \neq \perp$, and $u.TR.bval1 = s.TR2.cval1 \oplus s.TR2.inval2(Trans)(1 - s.TR2.inval(Rec))$.
That is, the high-level $bval1$ value is calculated as the $\oplus$ of the low-level $cval1$ value and the transmitter's input bit.

(i) $u.TR.bval = s.TR2.bval$.

(j) $u.Src_{tdpp} = s.Src_{tdpp}$.

(k) $u.Src_{yval} = s.Src_{yval}$.

(l) $u.Src_{bval1}.chosenval = s.Src_{cval1}.chosenval$.

(m) $u.Adv' = s.Adv'$.

(n) $u.Env = s.Env$.

(o) $u.TR.received_{tdp} = TR2.received_{tdp}$.

(p) $u.TR.received_z = TR2.received_z$.

(q) $u.TR.received_b = TR2.received_b$.

2. For ϵ_1 and ϵ_2, if for every $u \in supp(lstate(\epsilon_2))$ $u.TR.bval = \perp$, then one of the following holds:

   (a) For every $s \in support(lstate(\epsilon_1))$ and $u \in support(lstate(\epsilon_2))$, $s.Src_{cval1}.chosenval = u.Src_{bval1}.chosenval = \perp$.

   (b) For every $s \in support(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$, $u.TR.bval1 = s.TR2.cval1 = \perp$ and $lstate(\epsilon_1)$ projected on $Src_{cval1}.chosenval$ and $lstate(\epsilon_2)$ projected on $Src_{bval1}.chosenval$ is the uniform distribution on $\{0, 1\}$.

   (c) $lstate(\epsilon_1)$ projected on $TR2.cval1$ is the uniform distribution on $\{0, 1\}$ and $lstate(\epsilon_2)$ projected on $TR.bval1$ is the uniform distribution on $\{0, 1\}$.

10.6.2 The mapping proof

Lemma 10.19 *The relation R defined in Section 10.6.1 is a simulation relation from $Int2\|Env$ to $SIS\|Env$. Furthermore, for each step of $Int2\|Env$, the step correspondence yields at most three steps of $SIS\|Env$, that is, there is a mapping $corrtasks$ that can be used with R such that, for every ρ, T, $|corrtasks(\rho, T)| \leq 3$.*

Proof. We prove that R is a simulation relation from $RS\|Env$ to $Int1\|Env$ using the mapping $corrtasks : R^*_{Int2\|Env} \times R_{Int2\|Env} \rightarrow R^*_{SIS\|Env}$, which is defined as follows:

For any $(\rho, T) \in (R^*_{Int2\|Env} \times R_{Int2\|Env})$:

- If $T \in \{\{choose - rand_{tdpp}\}, \{rand(*)_{tdpp}\}, \{choose - rand_{yval}\}, \{rand(*)_{yval}\}, \{out(*)_{Rec}\}, \{out'(*)_{Rec}\}, \{out''(*)_{Rec}\}$ $\{send(1, *)_{Trans}\}, \{send(2, *)_{Rec}\}, \{send(3, *)_{Trans}\}\}, \{fix - bval_{Trans}\}\}$ then $corrtasks(\rho, T) = T$.
- If T is an output or internal task of Env or Adv, then $corrtasks(\rho, T) = T$.
- If $T = \{choose - rand_{cval1}\}$ then $corrtasks(\rho, T) = \{choose - rand_{bval1}\}$.
- If $T = \{rand(*)_{cval1}\}$ then $corrtasks(\rho, T) = \{rand(*)_{bval1}\}$.

We show that R satisfies the two conditions in Lemma 3.31.
Start condition: It is obvious that the Dirac measures on executions consisting of the unique start states s and u of $Int2 \| Env$ and $SIS \| Env$, respectively, are R-related. Property 1 holds because the state components of s and u on which R depends are all $\perp$. Property 2 holds because $s.Src_{cval1}.chosenval = u.Src_{bval1}.chosenval = \perp$.

Step condition: Suppose $(\epsilon_1, \epsilon_2) \in R$, $\rho_1 \in R*_{Int2\|Env}$, ϵ_1 is consistent with ρ_1, ϵ_2 is consistent with $full(corrtasks)(\rho_1)$,and T is a task of $Int2 \| Env$. Let $\epsilon_1' = apply(\epsilon_1, T)$ and $\epsilon_2' = apply(\epsilon_2, corrtasks(\rho_1, T))$.
Claim 1: Claim 1:

1. The state of Env is the same in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Let q_{Env} denote this state of Env.

   This follows from Property 1n.

2. The state of Adv' is the same in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Let q_{Adv} denote this state of Adv'.

   This follows from Property 1m.

Claim 2:

1. If T is an output or internal task of Env, then T is either enabled or disabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$ (simultaneously). Furthermore, if T is enabled in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, then:

   (a) There is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.

   (b) There is a unique transition of Env from q_{Env} with action a; let $tr_{Env} = (q_{Env}, a, \mu_{Env})$ be this transition.

2. If T is an output or internal task of Adv, then T is either enabled or disabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$ (simultaneously). Furthermore, if T is enabled in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, then:

   (a) There is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.

   (b) There is a unique transition of Adv from q_{Adv} with action a; let $tr_{Adv} = (q_{Adv}, a, \mu_{Adv})$ be this transition.

1. $T = \{choose - rand_{tdpp}\}$.
   We first show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Fix any pair of states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$. Task $T = corrtasks(\rho_1, T)$ is enabled in s (resp. u) iff $s.Src_{tdpp}.chosenval = \perp$ (resp. $u.Src_{tdpp}.chosenval = \perp$). Property 1j implies that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, as needed.

   (a) T is disabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   Let I be the singleton index $\{1\}$, let p be the Dirac measure on 1, and let $\epsilon_{11}' = \epsilon_1'$ and $\epsilon_{21}' = \epsilon_2'$. By Definition 3.3 we have $\epsilon_1' = \epsilon_1$, $\epsilon_2' = \epsilon_2$. Since $\epsilon_1 \; R \; \epsilon_2$, we have $\epsilon_{11}' \; R \; \epsilon_{21}'$, as needed. The trace distribution equivalence condition $tdist(\epsilon_1') = tdist(\epsilon_2')$ also holds since $tdist(\epsilon_1) = tdist(\epsilon_2)$.

   (b) T is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   We define the probability measures needed to show the step correspondence. Let p be the uniform probability measure on the index set $I = \{1 \cdots r\}$ where $r = |Tdp|$. That is, $p(j) = 1/r$ for each $j \in I$. For each $j \in I$, we define probability measure ϵ_{1j}' as follows. The support $supp(\epsilon_{1j}')$ is the set of executions $\alpha \in supp(\epsilon_1')$ such that $lstate(\alpha).Src_{tdpp}.chosenval$ is the jth element in domain Tdp. For each $\alpha \in supp(\epsilon_{1j}')$ of the form $\alpha' \, choose - rand_{tdpp} \, q$, let $\epsilon_{1j}'(\alpha) = \epsilon_1(\alpha')$. We define ϵ_{2j}' analogously from ϵ_2'.

Now fix $j \in I$; we show that $(\epsilon'_{1j}, \epsilon'_{2j}) \in R$. To do this, we establish Property 1 of R for ϵ'_{1j} and ϵ'_{2j}, and show trace distribution equivalence for ϵ'_{1j} and ϵ'_{2j}.

To establish Property 1, consider any states $s' \in supp(lstate(\epsilon'_{1j}))$ and $u' \in supp(lstate(\epsilon'_{2j}))$. By definitions of ϵ'_{1j} and ϵ'_{2j}, we know that $u'.Src_{tdpp}.chosenval = s'.Src_{tdpp}.chosenval$. Hence, Property 1j holds. Since no component other than $Src_{tdpp}.chosenval$ is updated by the application of T, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.

To establish Property 2, we need to show that for ϵ'_{1j} and ϵ'_{2j}, if for every $u' \in supp(lstate(\epsilon'_{2j}))$ $u'.TR.bval = \perp$, then one of the following holds:

i. For every $s' \in support(lstate(\epsilon_{1j}))$ and $u' \in supp(lstate(\epsilon'_{2j}))$, $s'.Src_{cval1}.chosenval = u'.Src_{bval1}.chosenval = \perp$.

ii. For every $s' \in support(lstate(\epsilon'_{1j}))$ and $u' \in supp(lstate(\epsilon'_{2j}))$, $u'.TR.bval1 = s'.TR2.cval1 = \perp$ and $lstate(\epsilon'_{1j})$ projected on $Src_{cval1}.chosenval$ and $lstate(\epsilon'_{2j})$ projected on $Src_{bval1}.chosenval$ is the uniform distribution on $\{0, 1\}$.

iii. $lstate(\epsilon'_{1j})$ projected on $TR2.cval1$ is the uniform distribution on $\{0, 1\}$ and $lstate(\epsilon'_{2j})$ projected on $TR.bval1$ is the uniform distribution on $\{0, 1\}$.

Suppose for every $u' \in supp(lstate(\epsilon'_{2j}))$ $u'.TR.bval = \perp$. Let u be any state in $supp(lstate(\epsilon_2))$ such that $u' \in supp(\mu_u)$ where $(u, a, \mu_u) \in D_{Int2 \| Env}$. Since T does not update $TR.bval$ and by Property 1i, we know that for every $u \in supp(lstate(\epsilon_2))$ $u.TR.bval = \perp$. Then by Property 2 one of the following is true.

i. For every $s \in support(lstate(\epsilon_1))$ and $u \in support(lstate(\epsilon_2))$, $s.Src_{cval1}.chosenval = u.Src_{bval1}.chosenval = \perp$.

ii. For every $s \in support(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$, $u.TR.bval1 = s.TR2.cval1 = \perp$ and $lstate(\epsilon_1)$ projected on $Src_{cval1}.chosenval$ and $lstate(\epsilon_2)$ projected on $Src_{bval1}.chosenval$ is the uniform distribution on $\{0, 1\}$.

iii. $lstate(\epsilon_1)$ projected on $TR2.cval1$ is the uniform distribution on $\{0, 1\}$ and $lstate(\epsilon_2)$ projected on $TR.bval1$ is the uniform distribution on $\{0, 1\}$.

We can show that if (a) holds for ϵ_1 and ϵ_2, then since T modifies neither $Src_{bval1}.chosenval$ nor $Src_{cval1}.chosenval$ Property (a) holds for ϵ'_{1j}, and ϵ'_{2j} as well. If Property (b) holds for ϵ_1 and ϵ_2, recall that we have defined ϵ'_{1j} in such a way that for each $\alpha \in supp(\epsilon'_{1j})$, where α is of the form $\alpha' \, a \, q_j$, we have $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$. Since T transitions modify neither $TR2.cval1$, $TR.bval1$, $Src_{bval1}.chosenval$, nor $Src_{cval1}.chosenval$ (b) holds for s', u', ϵ'_{1j}, and ϵ'_{2j}. If (c) holds for ϵ_1 and ϵ_2, we can argue as in the case for (b) to show that (c) holds for ϵ'_{1j} and ϵ'_{2j}.

2. $T = \{rand(*)_{tdpp}\}$.
We first show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. This part of the proof is identical to the case where $T = choose - rand_{tdpp}$.

   (a) T is disabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   This part of the proof is identical to the case where $T = choose - rand_{tdpp}$.

   (b) T is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   We show that there is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. We know by Property 1j that the state of Src_{tdpp} is the same in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Let q denote this state of Src_{tdpp}. By the next-action determinism property for Src_{tdpp} we know that there is a unique action $a \in T$ that is enabled in q. Since T is an output task of Src_{tdpp}, a is also the unique action in T that is enabled in each state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.

   The probability measures for this case are trivial: Let I be the singleton index set $\{1\}$, let p be the Dirac measure on 1, and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$. To show that $(\epsilon'_1, \epsilon'_2) \in R$, we establish Properties 1 and 2 of R for ϵ'_1 and ϵ'_2, and show trace distribution equivalence for ϵ'_1 and ϵ'_2.

To establish Property 1, consider any states $s' \in supp(lstate(\epsilon'_1))$ and $u' \in supp(lstate(\epsilon'_2))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, a, \mu_s) \in D_{Int2 \| Env}$. Similarly, let u be any state in $supp(lstate(\epsilon_2))$ such that $u' \in supp(\mu_u)$ where $(u, a, \mu_u) \in D_{Int2 \| Env}$.

By definitions of *Int2* and *SIS* we know that application of T updates *TR2.tdpp* in the *Int2* system, and *TR.tdpp* in the *SIS* system. We know by Property 1e that $u.TR2.tdpp = s.TR.tdpp$. By the effects of T in *TR1* and *TR2*, we know that $u'.TR2.tdpp = s'.TR.tdpp$; hence, Property 1e holds. Since no component other than *TR1.tdpp* in the *Int2* system and *TR.tdpp* in the *SIS* system is updated by the application of T, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.

To establish Property 2, we proceed as in the case for $T = choose - random_{tdpp}$.

3. $T = \{choose - rand_{yval}\}$.
   This case is analogous to the case where $T = \{choose - rand_{tdpp}\}$. In the argument for enabling we use Property 1k instead of Property 1j. In showing the step correspondence, we use the domain $\{0, 1\} \to D$ instead of Tdp and also use Property 1k instead of Property 1j.

4. $T = \{rand(*)_{yval}\}$.
   This case is analogous to the case where $T = \{rand(p)_{tdpp}\}$. In the argument for enabling we use Property 1k instead of Property 1j. In showing the step correspondence, we use the domain $\{0, 1\} \to D$ instead of Tdp and also use Property 1f instead of Property 1e.

5. $T = \{choose - random_{cval1}\}$.
   We show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1))$. We know by Property 1l that $Src_{cval1}.chosenval$ has the same value in all states in $supp(lstate(\epsilon_1))$. If $Src_{cval1}.chosenval = \perp$ in for each $s \in supp(lstate(\epsilon_1))$ then T is enabled, otherwise it is disabled.

   (a) T is disabled in $supp(lstate(\epsilon_1))$.
   This implies for each $s \in supp(lstate(\epsilon_1))$ we have $s.Src_{cval1}.chosenval \neq \perp$. We use Property 1l to show that the corresponding task sequence $\{choose - rand_{bval1}\}$ is also disabled. The rest of the proof is identical to the case where $T = \{choose - rand_{tdpp}\}$.

   (b) T is enabled in $supp(lstate(\epsilon_1))$.
   This implies for each $s \in supp(lstate(\epsilon_1))$ we have $s.Src_{cval1}.chosenval = \perp$. We use Property 1l to show that the corresponding task sequence $\{choose - rand_{bval1}\}$ is also enabled. Let I be the singleton index $\{1\}$, let p be the Dirac measure on 1, and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$.
   In showing $(\epsilon_{11}, \epsilon'_{21}) \in R$, Property 1 is easy to show, since all we do is update *Src.cval*1.*chosenval* and*Src.bval*1.*chosenval* to a non-$\perp$ value. Property 1l is preserved.
   For Property 2, suppose for every $u' \in supp(lstate(\epsilon'_{21}))$ $u'.TR.bval = \perp$. Let u be any state in $supp(lstate(\epsilon_2))$ such that $u' \in supp(\mu_u)$ where $(u, a, \mu_u) \in D_{Int2 \| Env}$. Since T does not update *TR.bval* and by Property 1i, we know that for every $u \in supp(lstate(\epsilon_2))$ $u.TR.bval = \perp$. Then by Property 2, we know that 2(a) holds for ϵ_1 and ϵ_2 since $\{choose - rand_{cval1}$ is enabled in $supp(lstate(\epsilon_1))$. Now, we can show that 2(b) holds, since the application of $\{choose - rand_{cval1}\}$ to ϵ_1 results in a distribution such that $lstate(\epsilon'_1)$ projected on $Src_{cval1}.chosenval$ is the uniform distibution on $\{0, 1\}$. Similarly, application of $\{choose - rand_{bval1}\}$ to $\epsilon_2$results in a distibution such that $lstate(\epsilon'_2)$ projected on $Src_{cval1}.chosenval$ is the uniform distibution on $\{0, 1\}$.

6. $T = \{rand(*)_{cval1}\}$.
   We show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1))$. We know by Property 1l that $Src_{cval1}.chosenval$ has the same value in all states in $supp(lstate(\epsilon_1))$. If $Src_{cval1}.chosenval = \perp$ in for each $s \in supp(lstate(\epsilon_1))$ then T is enabled, otherwise it is disabled.

(a) T is disabled in $supp(lstate(\epsilon_1))$.
This implies for each $s \in supp(lstate(\epsilon_1))$ we have $s.Src_{cval1}.chosenval = \perp$. We use Property 1l to show that the corresponding task sequence is also disabled. The rest of the proof is identical to the case where $T = choose - random_{tdpp}$.

(b) T is enabled in $supp(lstate(\epsilon_1))$.
This implies for each $s \in supp(lstate(\epsilon_1))$ we have $s.Src_{cval1}.chosenval \neq \perp$. We use Property 1l to show that the corresponding task sequence is also enabled.

Let I be the singleton index $\{1\}$, let p be the Dirac measure on 1, and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$.

In showing $(\epsilon_{11}, \epsilon'_{21}) \in R$, we have to consider Properties 1h and 2. To show Property 1h, consider any $u' in\, supp(lstate(\epsilon'_{21}))$ and suppose $u'.TR.bval \neq \perp$. Let u be any state in $supp(lstate(\epsilon_2))$ such that $u' \in supp(\mu_u)$ where $(u, a, \mu_u) \in D_{SIS \| Env}$. By the effects of $rand(b)_{bval1}$ transitions we know that for any $u.TR.bval \neq \perp$. By Property 1h we have that for any $s \in supp(lstate(\epsilon_1))$, $s.TR2.cval1 \neq \perp$, $s.TR2.inval(Trans) \neq \perp$, $s.TR2.inval(Rec)) \neq \perp$, and $u.TR.bval1 = s.TR2.cval1 \oplus s.TR2.inval2(Trans)(1 - s.TR2.inval(Rec))$. Since the $rand(b)_{bval1}$ and $rand(c)_{cval1}$ update $TR.bval1$ and $TR2.cval1$ only if they are $\perp$, we infer that $TR.bval1$ and $TR2.cval1$ are unchanged by the application of $\{rand(b)_{bval1}\}$ and $\{rand(c_{cval1}\}$. So, Property 1h continues to hold.

For Property 2, suppose for every $u' \in supp(lstate(\epsilon'_{21}))$ $u'.TR.bval = \perp$. Let u be any state in $supp(lstate(\epsilon_2))$ such that $u' \in supp(\mu_u)$ where $(u, a, \mu_u) \in D_{SIS \| Env}$. Since T does not update $TR.bval$ and by Property 1i, we know that for every $u \in supp(lstate(\epsilon_2))$ $u.TR.bval = \perp$. Then by Property 2, we know that either 2(b) or 2(c) holds for ϵ_1 and ϵ_2, since $\{random(c)_{cval1}\}$ is enabled. If 2(b) holds for ϵ_1 and ϵ_2, then we can show that 2(c) holds for ϵ'_1 and ϵ'_2. That is, if a value has been chosen but not output yet in Src_{cval1} and Src_{bval1}, the application of $\{random(c)_{cval1}\}$ and $\{random(c)_{bval1}\}$, updates $TR2.cval1$ and $TR.bval1$ so that $lstate(\epsilon'_{11})$ projected on $TR2.cval1$ is the uniform distribution on $\{0, 1\}$ and $lstate(\epsilon'_{21})$ projected on $TR.bval1$ is the uniform distribution on $\{0, 1\}$. If 2(c) holds, we know that the projection of $lstate(\epsilon_1)$ and $lstate(\epsilon_2)$ on $TR2.bval1$ and $TR.cval1$ are uniform distibutions of $\{0, 1\}$. That is, the chosen values have already been output. Hence, the application of $\{random(c)_{cval1}\}$ and $\{random(c)_{bval1}\}$ have no effect and 2(c) holds for ϵ'_1 and ϵ'_2.

7. $T = \{fix - bval_{Trans}\}$.
By definition of R, Properties 1f, 1g, 1b, 1c, 1d, 1i, 1p, 1h and 2, we know that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1))$.
We examine the following two cases:

(a) T is disabled in $supp(lstate(\epsilon_1))$.
We show that T is uniformly disabled in $supp(lstate(\epsilon_2))$. Fix any $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$. If $s.TR2.yval = \perp$ the by Property 1f $u.TR.yval = \perp$. If $s.TR2.zval = \perp$ the by Property 1g $u.TR.zval = \perp$. If $s.TR2.cval1 = \perp$ then by Property 1h, $u.TR.bval = \perp$. Since $s.TR2.cval1 = \perp$ ither 2(a) or 2(b) holds, each of which implies that $TR.bval1 = \perp$. If $s.TR2.inval(Trans) = \perp$ then by Property 1c $u.TR.inval(Trans) = \perp$. If $s.TR2.inval2(Trans) = \perp$ then by Property 1b $u.Funct.inval(Trans) = \perp$, and by Lemma 10.2, $u.TR.inval(Trans) = \perp$. If $s.TR2.inval(Rec) = \perp$ then by Property 1d $u.TR.inval(Rec) = \perp$. $s.TR2.received_z = \perp$ then by Property 1p $u.TR.received_z = \perp$. Finally if $s.TR2.bval \neq \perp$ the by Property 1i $u.TR.bval \neq \perp$. Hence, T is uniformly disabled in $supp(lstate(\epsilon_2))$.

Let I be the singleton index $\{1\}$, let p be the Dirac measure on 1, and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$. By Definition 3.3 we have $\epsilon'_1 = \epsilon_1$, $\epsilon'_2 = \epsilon_2$. Since $\epsilon_1\ R\ \epsilon_2$, we have $\epsilon'_{11}\ R\ \epsilon'_{21}$, as needed. The trace distribution equivalence condition $tdist(\epsilon'_1) = tdist(\epsilon'_2)$ also holds since $tdist(\epsilon_1) = tdist(\epsilon_2)$.

(b) T is enabled in $supp(lstate(\epsilon_1))$.
By precondition of $fix-bval_{Trans}$, we know that for any $s \in supp(lstate(\epsilon_1))$, $s.TR2.yval \neq \perp$, $s.TR2.cval1 \neq \perp$, $s.TR2.inval(Trans) \neq \perp$, $s.TR2.inval2(Trans) \neq \perp$, $s.TR2.inval(Rec) \neq \perp$, $s.TR2.received_z \neq \perp$, $s.TR2.bval = \perp$. Let u be any state in $supp(lstate(\epsilon_2))$. By Properties 1f, refR: case2: int2-sis: 1g 1c, 1d, 1i, 1p, we know that $u.TR2.yval \neq \perp$, $u.TR2.zval \neq \perp$, $u.TR2.inval(Trans) \neq \perp$, $u.TR2.inval(Rec) \neq \perp$, $u.TR2.received_z \neq \perp$, $u.TR2.bval = \perp$. Since $u.TR2.bval = \perp$ and $s.TR2.cval1 \neq \perp$, we know that Property 2(c) holds and therefore, $lstate(\epsilon_2))$ projected onto $TR.bval1$ is the uniform distribution on $\{0,1\}$. Hence $u.TR.bval1 \neq \perp$ and T is enabled in $supp(lstate(\epsilon_1))$.

Next, we define the probability measures. Let p be the uniform probability measure on the index set $I = \{1,2\}$. That is, $p(j) = 1/2$ for each $j \in I$. For each $j \in I$, we define probability measure ϵ'_{1j} as follows. The support $supp(\epsilon'_{11})$ is the set of executions $\alpha \in supp(\epsilon'_1)$ such that $lstate(\alpha).TR2.bval(1 - inval(Rec)) = 0$ and the support $supp(\epsilon'_{12})$ is the set of execution fragments $\alpha \in supp(\epsilon'_1)$ such that $lstate(\alpha).TR2.bval(1 - inval(Rec)) = 1$. For each $\alpha \in supp(\epsilon'_{1j})$ of the form $\alpha'\ fix - bval_{Trans}\ q$, let $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$. We define ϵ'_{2j} analogously from ϵ'_2.

Now fix $j \in I$; we show that $(\epsilon'_{1j}, \epsilon'_{2j}) \in R$. To do this, we establish Property 1 of R for ϵ'_{1j} and ϵ'_{2j}, and show trace distribution equivalence for ϵ'_{1j} and ϵ'_{2j}.

To establish Property 1, consider any states $s' \in supp(lstate(\epsilon'_{1j}))$ and $u' \in supp(lstate(\epsilon'_{2j}))$. By definitions of ϵ'_{1j} and ϵ'_{2j}, Properties 1f, 1c, 1d, and Definition 3.3, we know that $u'.TR.bval = s'.TR2.bval$.

Hence, Property 1i holds. We also need to show that Property 1h holds. This is immediate from the effects of T in $TR2$ and TR, and the definitions of ϵ'_{1j} and ϵ'_{2j}. Since no component other than $bval$ is updated by the application of T, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_{1j} and ϵ'_{2j}.

Since we know that for every $u' \in supp(lstate(\epsilon'_{2j}))$, $u'.TR.bval \neq bot$, Property 2 trivially holds.

8. $T = \{fix - zval_{Rec}\}$.
We first show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Fix any pair of states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$. First, assume that T is enabled in s. This implies that $s.TR2.yval \neq \perp$, $s.TR2.inval(Rec) \neq \perp$, $s.TR2.tdpp \neq \perp$, $s.TR2.received_{tdp} \neq \perp$, and $s.TR2.zval = \perp$. Since $s.TR2.yval \neq \perp$, by Property 1f we have $u.TR.yval \neq \perp$. Since $s.TR2.inval \neq \perp$, by Property 1d, we have $u.TR.inval(Rec) \neq \perp$. Since $s.TR2.tdpp \neq \perp$, by Property 1e we have $u.TR1.tdpp \neq \perp$. Since $s.TR2.received_{tdp} \neq \perp$, by Property 1o we have $u.TR.received_{tdp} \neq \perp$. Since $s.Rec.zval = \perp$, by Property 1g we have $u.TR1.zval = \perp$. Hence, T is enabled in u, as needed.

Now assume that T is disabled in s. We need to show that if any of the preconditions of $fix - zval_{Rec}$ is false in s then at least one of the preconditions in u is false. If $s.TR2.yval = \perp$ then by Property 1f, we have $u.TR.yval = \perp$. If $s.TR2.inval(Rec) = \perp$, then by Property 1d, we have $u.TR.inval(Rec) = \perp$. If $s.TR2.tdpp = \perp$, then by Property 1e, we have $u.TR.tdpp = \perp$. If $s.TR2.receieved_{tdp} = \perp$, then by Property 1o $u.TR.received_{tdpp} = \perp$. If $s.TR2.zval \neq \perp$, by Property 1g we have $u.TR1.zval \neq \perp$. Hence T is disabled in u, as needed.

(a) T is disabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
This part of the proof is identical to the case where $T = choose - rand_{tdpp}$.

(b)

(c) T is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
The rest of the proof is easy because $zval$ is computed in the same way in both the $TR2$ and SIS systems. We let p be the Dirac measure in the singleton index set $\{1\}$, and use Property 1g.

9. $T = \{out'(*)_{Rec}\}$.
T is output from *Funct* to *TR2* in *Int2* and from *Funct* to *TR* in SIS. We can show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, by using Property 1a.

   (a) T is disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2)))$.
   Identical to the corresponding part in the case for $T = choose - rand_{tdpp}$.

   (b) T is enabled $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2)))$.
   We can let p be the Dirac measure on the singleton index set $\{1\}$, and use Properties 1c and 1a to show that Property 1 holds. To show Property 2, we proceed as in case for $T = choose - rand_{tdpp}$.

10. $T = \{out''(*)_{Rec}\}$.
T is output from *TR2* to Adv' in *Int2* and from *TR* to Adv' in SIS. We can show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, by using Properties 1c and 1q.

   (a) T is disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2)))$.
   Identical to the corresponding part in the case for $T = choose - rand_{tdpp}$.

   (b) T is enabled $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2)))$.
   We can let p be the Dirac measure on the singleton index set $\{1\}$, and use Property 1m to show that Property 1 holds. To show Property 2, we proceed as in case for $T = choose - rand_{tdpp}$.

11. $T = \{send(1, *)_{Trans}\}$.
We first show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. This is analogous to case for $T = choose - rand_{tdpp}$. We use Property 1e to show that $TR.tdpp = TR2.tdpp$ for all states in $supp(lstate(\epsilon_1))$ and $supp(lstate(\epsilon_2))$.

   (a) T is disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   Identical to the corresponding part in the case for $T = choose - rand_{tdpp}$.

   (b) T is enabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   First, we show that there is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. We know by Property 1e that variables *Trans.tdpp* and *TR1.tdpp* have the same unique value in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Since the parameter f in $send(1, f)_{Trans}$ is defined to be $TR2.tdpp.funct$ we conclude that the action $send(1, TR2.tdpp.funct)$ is the unique action in T that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. We use a as a shorthand for $send(1, TR2.tdpp.funct)$ in the rest of the proof for this case.

   Let I be the singleton index set $\{1\}$, let p be the Dirac measure on 1, and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$. To show that $(\epsilon'_1, \epsilon'_2) \in R$, we establish Property 1 of R for ϵ'_1 and ϵ'_2, and show trace distribution equivalence for ϵ'_1 and ϵ'_2. To establish Property 1, consider any state $s' \in supp(lstate(\epsilon'_1))$ and $u' \in supp(lstate(\epsilon'_2))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, a, \mu_s) \in D_{Int2 \| Env}$. Similarly, let u be any state in $supp(lstate(\epsilon_2))$ such that $u' \in supp(\mu_u)$ where $(u, a, \mu_u) \in D_{SIS \| Env}$.

   By definitions of *Int2* and *SIS* we know that application of T updates only $Adv'.messages$ in the *Int2* system and $Adv'.messages$ in the *TR* system. By Property 1m, $u.Adv' = s.Adv'$. It is obvious that $u'.Adv' = s'.Adv$ and that Property 1m holds. Since no component other than $Adv'.messages$ is updated, we conclude that Property 1 holds.

   To show Property 2, we proceed as in case for $T = choose - rand_{tdpp}$.

   The fact that $tdist(\epsilon'_1) = tdist(\epsilon'_2)$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_1 and ϵ'_2.

12. $T = \{send(2, *)_{Rec}\}$.
We first show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. This

is analogous to case for $T = choose - rand_{tdpp}$. We use Property 1g to show that $TR2.zval = TR.zval$ for all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.

(a) T is disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
Identical to the corresponding part in the case for $T = choose - rand_{tdpp}$.

(b) T is disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
Next, we show that there is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. We know by Property 1g that variables $TR2.zval$ and $TR.zval$ have the same unique value in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, and there is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Note that here a is $send(2, z)_{Rec}$ for a fixed value of z.
The rest is identical to the proof for $T = \{send(1, *)_{Trans}\}$.

13. $T = \{send(3, *)_{Trans}\}$.
We first show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. This is analogous to case for $T = choose - random_{tdpp}$. We use Property 1i to show that $TR2.bval = TR.bval$ for all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.

(a) T is disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
Identical to the corresponding part in the case for $T = choose - rand_{tdpp}$.

(b) T is enabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
We show that there is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, arguing as in the case for $T = \{send(1, f)_{Trans}\}$. Here, the unique action is determined by fixing the value of parameter b to the value of variables $TR2.bval$ and $TR.bval$, which is the same in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. The rest of the proof is identical to the proof for $T = \{send(1, f)_{Trans}\}$.

14. $T = \{out'(*)_{Rec}\}$.
T is output from *Funct* to *TR2* in the *Int2* system and from *Funct* to *TR* in the SIS system. We first show that T is uniformly enabled or disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. This is analogous to case for $T = choose - rand_{tdpp}$. We use Property 1a.

(a) T is disabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
Identical to the corresponding part in the case for $T = choose - rand_{tdpp}$.

(b) T is enabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
We first show that there is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, arguing as in the case for $T = \{send(1, f)_{Trans}\}$. Let I be the singleton index set $\{1\}$, let p be the Dirac measure on 1, and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$. To show that $(\epsilon'_1, \epsilon'_2) \in R$, we establish Property 1 of R for ϵ'_1 and ϵ'_2, and show trace distribution equivalence for ϵ'_1 and ϵ'_2. To establish Property 1, we use Property 1c. To show Property 2, we proceed as in the case for $T = choose - rand_{tdpp}$.

15. T is a task of *Env*.
Claim 2 implies that T is uniformly enabled or disabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. If T is disabled, we simply define $I = \{1\}$, $\epsilon'_{11} = \epsilon_1$, $\epsilon'_{21} = \epsilon_2$, and we have that $\epsilon'_{11}\ R\ \epsilon'_{21}$ since $\epsilon_1\ R\ \epsilon_2$. Suppose now that T is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Claim 2 now implies that there is a unique action a enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$ and that there is a unique transition of *Env* from q_{Env} with action a; let $tr_{Env} = (q_{Env}, a, \mu_{Env})$ be this transition. We distinguish several cases, according to the possible values for a.

(a) a is an input action of *TR2* and *Funct* in *Int2* and only *Funct* in SIS. This means that $a = in(x)_{Trans}$ for some fixed x.

We define the probability measures needed to show the step correspondence. Suppose that $supp(\mu_{Env})$ is the set $\{q_j : j \in I\}$ of states of Env, where I is a countable index set. Let p be the probability measure on the index set I such that, for each $j \in I$, $p(j) = \mu_{Env}(q_j)$. For each $j \in I$, we define probability measure ϵ'_{1j} as follows. The support $supp(\epsilon'_{1j})$ is the set of executions $\alpha \in supp(\epsilon'_1)$ such that $lstate(\alpha).Env = q_j$. For each $\alpha \in supp(\epsilon'_{1j})$ of the form $\alpha'\ a\ q_j$, let $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$. We define ϵ'_{2j} analogously from ϵ'_2.

Now fix $j \in I$; it remains to show that $(\epsilon'_{1j}, \epsilon'_{2j}) \in R$. To do this, we establish Property 1 of R for ϵ'_{1j} and ϵ'_{2j}, and show trace distribution equivalence for ϵ'_{1j} and ϵ'_{2j}.

To establish Property 1, consider any states $s' \in supp(lstate(\epsilon'_{1j}))$ and $u' \in supp(lstate(\epsilon'_{2j}))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, a, \mu_s) \in D_{Int2 \| Env}$. Similarly, let u be any state in $supp(lstate(\epsilon_2))$ such that $u' \in supp(\mu_u)$ where $(u, a, \mu_u) \in D_{SIS \| Env}$.

By Property 1a, $s.Funct = u.Funct$. By the effects of $in(x)_{Trans}$, we have $s'.Funct = u'.Funct$. By Property 1b $u.Funct.inval(Trans) = s.TR2.inval2(Trans)$. By the effects of $in(x)_{Trans}$, we have $u'.Funct.inval(Trans) = s'.TR2.inval2(Trans)$. We also know that 1n holds by definition of ϵ'_{1j} and ϵ'_{2j}. Since no component other than $Funct.inval(Trans)$, $TR.inval2(Trans)$ and Env are updated, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.

We can show that Property 2 holds as in the case for $choose - rand_{tdpp}$.

The fact that $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_{1j} and ϵ'_{2j}.

(b) a is an input action of $TR2$, $Funct$, and Adv' in $Int2$, and TR, $Funct$, and Adv' in SIS. This means that $a = in(i)_{Rec}$ for some fixed i.

Here, a is shared between Env and Adv' in both systems. We must consider the probabilistic branching of Adv' as well as Env in this case.

Recall from Claim 1 that the state of Adv' is the same in all sates in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, and we let q'_{Adv} denote this state. Claim 2 states that there is a unique transition of Adv' with action a from q'_{Adv}. Let $tr'_{Adv} = (q'_{Adv}, a, \mu'_{Adv})$ be this transition.

Next we define the probability measures needed to show the step correspondence. Suppose that $supp(\mu_{Env} \times \mu'_{Adv})$ is the set $\{(q_{j1}, q_{j2}) : j \in I\}$ of pairs of states, where I is a countable index set. Let p be the probability measure on the index set I such that, for each $j \in I$, $p(j) = (\mu_{Env} \times \mu_{Adv})(q_{1j}, q_{2j})$. For each $j \in I$, we define probability measure ϵ'_{1j} as follows. The $supp(\epsilon'_{1j})$ is the set of executions $\alpha \in supp(\epsilon'_1)$ such that $lstate(\alpha).Env = q_{1j}$ and $lstate(\alpha).Adv' = q_{2j}$. For each $\alpha \in supp(\epsilon'_{1j})$ of the form $\alpha'\ a\ q$, let $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$. We construct ϵ'_{2j} analogously from ϵ'_2.

The rest of the proof for this case follows the proof for $a = in(x)_{Trans}$. The only difference is that in showing Property 1 for ϵ'_{1j} and ϵ'_{2j}, for a fixed j, we use the fact that application of T affects only $Funct.inval(Rec)$, $TR2.inval(Rec)$, Adv', and Env in the $Int2$ system, and $Funct.inval(Rec)$, $TR.inval(Rec)$, Adv', and Env in the $Int1$ system, and use Properties 1a, 1d, 1m, and 1n.

(c) a is an input of Adv' but not an input of $Funct$, $TR2$, or TR.

Claim 2 implies that there is a unique transition of Adv' with action a from q'_{Adv}. Let $tr'_{Adv} = (q'_{Adv}, a, \mu'_{Adv})$ be this transition.

Suppose that $supp(\mu_{Env} \times \mu_{Adv})$ is the set $\{(q_{j1}, q_{j2}) : j \in I\}$ of pairs of states, where I is a countable index set. Let p be the probability measure on the index set I such that, for each $j \in I$, $p(j) = (\mu_{Env} \times \mu_{Adv})(q_{1j}, q_{2j})$. For each $j \in I$, we define probability measure ϵ'_{1j} as follows. The support $supp(\epsilon'_{1j})$ is the set of executions $\alpha \in supp(\epsilon'_1)$ such that $lstate(\alpha).Env = q_{1j}$ and $lstate(\alpha).Adv = q_{2j}$. For each $\alpha \in supp(\epsilon'_{1j})$ of the form $\alpha'\ a\ q$, let $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$. We construct ϵ'_{2j} analogously from ϵ'_2.

In the rest of the proof we proceed as for $a = \{in(x)_{Trans}\}$. The only difference is that in showing Property 1 for ϵ'_{1j} and ϵ'_{2j}, for a fixed j, we use the fact that application of T affects only the states of Adv', and Env and use Properties 1m and 1m.

(d) a is an internal action of Env or an output action of Env that is not an input of $Funct$, $TR2$ or TR.

To show the step correspondence, we proceed as for $a = in(x)_{Trans}$. The only difference is that in showing Property 1 for ϵ'_{1j} and ϵ'_{2j}, for a fixed j, we use the fact that application of T affects only the state of Env, and use Property 1n.
For each index j in the decomposition, the fact that $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_{1j} and ϵ'_{2j}.

16. T is a task of Adv'.

Claim 2 implies that T is either simultaneously enabled or disabled in every state in $supp(lstate(\epsilon_1))\cup supp(lstate(\epsilon_2))$. If T is disabled, we simply define $I = \{1\}$, $\epsilon'_{11} = \epsilon_1$, $\epsilon'_{21} = \epsilon_2$, and we have that $\epsilon'_{11} R \epsilon'_{21}$ since $\epsilon_1 R \epsilon_2$. Suppose now that T is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. Claim 2 now implies that there is a unique action a enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$ and that there is a unique transition of Adv' fr'om q'_{Adv} with action a; let $tr'_{Adv} = (q_{Adv}, a, \mu'_{Adv})$ be this transition. We distinguish several cases, according to possible values for a.

(a) $a = \{out(x)_{Rec}\}$ for a fixed x.

Note that a does not acuse any braching in Adv' but it may cause branching in Env. Recall from Claim 1 that the state of Env is the same in all sates in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, and we let q_{Env} denote this state. Claim 2 states that there is a unique transition of Env with action a from q_{Env}. Let $tr_{Env} = (q_{Env}, a, \mu_{Env})$ be this transition.
To show the step correspondence, we proceed as for the case where T is a task of Env which consists of the action $a = in(x)_{Trans}$, decomposing the measures generated by the application of T according to the resulting state in Env, and using Property 1n to show that Property 1 holds for each component measure.
For each index j in the decomposition, the fact that $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$ follows from the fact that $tdist(\epsilon_1) = tdist(\epsilon_2)$ and the definitions of ϵ'_{1j} and ϵ'_{2j}.

(b) a is an input action of Env that is different from $out(x)$.
Claim 2 states that the state of Env is the same in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, let q_{Env} be this unique state. Also, there is a unique transition of Env with action a from q_{Env}. Let $tr_{Env} = (q_{Env}, a, \mu_{Env})$ be this transition.
To show the step correspondence, we proceed as in proof case where T is a task of Env and the unique action in this task is an input of Adv. using Properties 1m and 1n to show Property 1.

(c) a is an input action of TR and $TR2$.
This means that $a = receive(1, f)_{Rec}$ for some f, $a = receive(2, z)_{Trans}$ for some z, or $a = receive(3, b)_{Rec}$ for some f. Suppose first that $a = receive(1, f)_{Rec}$ for some f. The action a is an output of Adv' and input of $TR2$ in $Int2$ and TR in SIS.
The rest is similar to the proof for $T = \{send(1, f)_{Trans}\}$. The only difference is that in showing that Property 1 holds, we must show that Property 1o holds. We use the fact that application of T updates only $TR2.received_{tdp}$ in $Int2$ and $TR.received_{tdp}$ in SIS, and its effect is to set these variables to a non-$\perp$ value if they are $\perp$, and Property 1o can be easily seen to holds in this case.
Suppose now that $a = receive(2, z)_{Trans}$ for some z. The rest is similar to the proof for $T = \{send(1, f)_{Trans}\}$. The only difference is that in showing that Property 1 holds, we

must show that Property 1p holds. We use the fact that application of T updates only $TR2.received_z$ in *Int2* and $TR.received_z$ in SIS, and its effect is to set these variables to a non-$\perp$ value if they are $\perp$, and Property 1p can be easily seen to holds in this case.

Suppose now that $a = receive(3, b)_{Rec}$ for some b. The rest is similar to the proof for $T = \{send(1, f)_{Trans}\}$. The only difference is that in showing that Property 1 holds, we must show that Property 1q holds. We use the fact that application of T updates only $TR2.received_b$ in *Int2* and $TR.received_b$ in SIS, and its effect is to set these variables to a non-$\perp$ value if they are $\perp$, and Property 1q can be easily seen to holds in this case.

(d) a is either an output action of *Adv* that is not an input action of *Env*, *TR*, or *TR2*, or is an internal action of *Adv*.
To show the step correspondence, we proceed as for $a = in(x)_{Trans}$, in the case where T is an output task of *Env*, but using *Adv*′ instead of *Env*. In showing Property 1 for ϵ'_{1j} and ϵ'_{2j}, for a fixed j, we use the fact that application of T affects only the state of *Adv*′ and use Property 1m.

□

11 Conclusions

We believe that our task-PIOA model, which allows both nondeterministic and probabilistic system descriptions, and includes explicit treatment of computational issues provides a suitable framework for rigorous and systematic analysis of a wide range of cryptographic protocols. It allows one to use multiple levels of abstraction in doing the proofs and to isolate computational reasoning to certain stages in the analysis.

Our plans for the near future include application of our methods to the analysis of protocols that have more powerful adversaries (active rather than passive; adaptive). We also plan to establish general composition theorems in the style of the universal composition theorem of Canetti [Can01], and to use these results for the analysis of a key exchange protocol based on more sophisticated computational assumptions.

Acknowledgements We thank Silvio Micali for encouraging us to make a generalization in the definition of task-PIOAs that enhances the branching power of adversaries.

References

[BCT04] Gilles Barthe, Jan Cerderquist, and Sabrina Tarento. A machine-checked formalization of the generic model and the random oracle model. In D. Basin and M. Rusinowitch, editors, *Automated Reasoning: Second International Joint Conference, IJCAR 2004*, number 3097 in LNCS, pages 385–399, Cork, Ireland, 2004. Springer-Verlag.

[Bla05] Bruno Blanchet. A computationally sound mechanized prover for security protocols. Cryptology ePrint Archive, Report 2005/401, Nov. 2005. `http://eprint.iacr.org/`.

[BPW03] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proceedings 10th ACM conference on computer and communications security (CCS)*, 2003. Extended version at the Cryptology ePrint archive, `http://eprint.iacr.org/2003/015/`.

[BPW04a] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In M. Naor, editor, *First Theory of Cryptography Conference, TCC 2004*, volume 2951 of *LNCS*, pages 336–354, Cambridge, MA, USA, February 2004. Springer-Verlag.

[BPW04b] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Secure asynchronous reactive systems. Cryptology ePrint Archive, Report 2004/082, 2004. http://eprint.iacr.org/.

[BR04] Mihir Bellare and Phillip Rogaway. The game-playing technique and its application to triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004. http://eprint.iacr.org/.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *The Proceedings of 42nd FOCS*, 2001. Full version available at http://eprint.iacr.org/2000/067.

[CCK+05] Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Using probabilistic I/O automata to analyze an oblivious transfer protocol. Technical Report Technical Report MIT-LCS-TR-1001a, MIT CSAIL, 2005. This is the revised version of Technical Report MIT-LCS-TR-1001.

[EGL85] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *CACM*, 28(6):637–647, 1985.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the 19th Symposium on Theory of Computing (STOC)*, pages 218–229. ACM, 1987.

[Gol01] Oded Goldreich. *Foundations of Cryptography*, volume I Basic Tools. Cambridge Universirty Press, 2001. Reprint of 2003, page 64.

[Hal05] Shai Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181, 2005. http://eprint.iacr.org/.

[JL91] B. Jonsson and K.G. Larsen. Specification and refinement of probabilistic processes. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, pages 266–277, Amsterdam, July 1991.

[LMMS98] P.D. Lincoln, J.C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the 5th ACM conference on Computer and Communications Security (CCS-5)*, pages 112–121, San Francisco, 1998.

[LSV03] Nancy Lynch, Roberto Segala, and Frits Vaandrager. Compositionality for probabilistic automata. In *Proceedings of the 14th International Conference on Concurrency Theory*, volume 2761 of *LNCS*, pages 208–221, Marseille, France, September 2003. Springer. Fuller version appears in Technical Report MIT-LCS-TR-907, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, and was submitted for journal publication.

[MMS03] P. Mateus, J.C. Mitchell, and A. Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time calculus. In R. Amadio and D. Lugiez, editors, *Proceedings of CONCUR 2003 - Concurrency Theory*, number 2761 in LNCS, pages 327–349, Marseille, France, 2003. Springer.

[PSL00] A. Pogosyants, Roberto Segala, and Nancy Lynch. Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study. *Distributed Computing*, 13(3):155–186, 2000.

[PW00] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *7th ACM Conference on Computer and Communications Security*, pages 245–254, New York, 2000. ACM Press.

[PW01] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–200, Oakland, CA, May 2001. IEEE Computer Society.

[RMST04] Ajith Ramanathan, John Mitchell, Andre Scedrov, and Vanessa Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. In Igor Walukiewicz, editor, *Foundations of Software Science and Computation Structures: 7th International Conference, FOSSACS 2004*, number 2987 in LNCS, pages 468–483, Barcelona, Spain, 2004. Springer-Verlag.

[Seg95] Robert Segala. *Modeling and Verification of Randomized Distributed Real-time systems*. PhD thesis, Massachusetts Institute of Technology, 1995. Available as Technical Report MIT/LCS/TR-676.

[Seg97] Roberto Segala. Compositional verification of randomized distributed algorithms. In *The Proceedings of Compositionality – the significant difference*, volume 1536 of *LNCS*, pages 515–540, 1997.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/`.

[Tar05] S. Tarento. Machine-checked security proofs of cryptographic signature schemes. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *Computer Security – ESORICS 2005, 10th European Symposium on Research in Computer Security*, volume 3679 of *LNCS*, Milan, Italy, 2005. Springer.

A Comparison of task-PIOAs with existing frameworks

In this section we compare the task-PIOA framework with existing frameworks for modeling cryptographic protocols. In particular, we discuss the *Secure Asynchronous Reactive Systems* of [PW01, BPW04b] and the *Probabilistic Polynomial-Time Process Calculus* of [LMMS98, MMS03, RMST04].

A.1 Secure Asynchronous Reactive Systems

We first discuss the relations between task-PIOAs and another extension of PIOAs introduced by Backes, Pfitzmann and Waidner [PW01, BPW03, BPW04a, BPW04b].

A.1.1 System Types

The reactive systems in [PW01, BPW04b] are given by collections of *machines*. Each machine M is specified (in part) by a transition function of the following form:

$$\Delta : S \times I \to Disc(S \times O).$$

Here S is the state space and I and O denote the set of input and output signals, respectively. Each signal is a tuple of finite strings over a fixed finite alphabet Σ. Every tuple in I has a fixed arity, which is determined by the number of input ports associated with M. Similarly for O.

The transition function of a PIOA defined here has a very different form (cf. Section 3):

$$\Delta : S \times (I \cup O \cup H) \to (\{\bot\} + Disc(S)).$$

If $\Delta(s, a) = \bot$, then a is not enabled in s; otherwise, $\Delta(s, a)$ specifies a probability distribution on the resulting state after the execution of a.

We highlight three differences.

- Machines in [PW01, BPW04b] are of a decidedly *functional* character: given a tuple of input strings, some randomized computation is performed, producing a tuple of output strings. In contrast, the representation of a PIOA is entirely *operational*: the actions in $I \cup O \cup H$ are abstractions of activities of a system, rather than values manipulated by the system. Thus, in a PIOA, inputs and outputs need not correspond as tightly as they do in machines.

- Machines in [PW01, BPW04b] do not have internal/hidden transitions. This is because internal computations are abstracted away (provided the computation does not exceed certain limits on time and space).

- The only form of nondeterminism in a machine resides in the choices between inputs. In other words, nondeterminism can only be used to model uncertainties in the external environment. PIOAs, on the other hand, allow nondeterministic choices *within* a system (e.g., between different output actions enabled in the same state). Therefore, a closed system of machines is completely specified (up to coin tosses), whereas a closed system of PIOAs may contain many nondeterministic choices.

A.1.2 Communication

In [PW01, BPW04b], machines are divided into three classes: *simple machines*, *master schedulers* and *buffers*. These machines communicate with each other via *ports*, which are classified as: *simple*, *buffer* and *clock*.

Each buffer B specifies a high-level connection between a *unique* pair of non-buffer machines. Messages placed in B are delivered only when B is scheduled via its clock port. This scheduling is designated to a unique non-buffer machine (one that "owns" the clock port of B), which also determines the order in which messages in B are delivered.

Notice, both low-level (based on port names) and high-level (based on buffers) connections are "handshakes". That is, at most two machines are involved in a synchronization (barring the machine responsible for scheduling the buffer).

In the case of PIOAs, communications may be "broadcasts". Each action a is "owned" by a unique PIOA P (i.e., a is an output action of P), but any number of other PIOAs can have a as an input action. A hiding operation is available, which reclassifies certain output actions as hidden actions, thus preventing synchronization with PIOAs outside the scope of hiding.

Also, actions in our setting need not represent messages. They can be used to model synchronized steps of different processes.

A.1.3 Modeling

Channels In [PW01, BPW04b], three kinds of communication channels are considered: secure, authenticated and insecure.

By default, a communication channel between two machines is secure: no third party can access message contents. Another party might still be able to schedule the buffer for this channel. Authenticated channels are modeled by creating a copy of the out-port corresponding to the authenticated channel, and by connecting this new port to the adversary. Insecure channels are directly connected to the adversary.

In the case of PIOAs, the basic communication channels correspond to authenticated channels: every output action may correspond to several input actions, including adversarial ones. As for machines, insecure channels are modeled by routing messages through the adversary. We use two techniques to define secure channels: the first one consists in hiding output actions in the composition of the PIOAs corresponding to the sending and receiving ends of the channels, while the second one consists in specifying constraints on the signature of specific (adversarial) PIOAs to prevent them from synchronizing on the corresponding output actions.

In our analysis of the OT protocol, we consider a slightly different kind of communication channels: all protocol messages are routed through the adversary, but we define the adversary in such a way that it can only send messages it received before. In the [PW01, BPW04b] communication model, this would correspond to an authenticated channel with buffer scheduled by the adversary.

Composition In [PW01, BPW04b], *collections* of machines are defined as sets of machines, with the intuition that machines in a collection will interact. These interactions can be the transmission of a message (through a buffer) or the scheduling of buffers.

In the case of PIOAs, we can compose several PIOAs, yielding a new PIOA. This enables, for instance, defining the role of a protocol participant through several simple PIOAs, each of these PIOAs describing a different aspect of this protocol role. So, considering several PIOAs can also be used as a way to analyze systems in a more modular way.

A.1.4 Scheduling

In [PW01, BPW04b], scheduling is *distributed*: it is performed collectively by all machines via scheduling of buffers. The following algorithm is used.

(1) The current active machine M reads all of its inputs and carries out changes dictated by its transition function.

(2) All of the outputs produced in step (1) are copied to corresponding buffers.

(3) If in step (1) M schedules at least one buffer, then let B be the first such buffer and M' be the receiving machine of B. (This is possible because an ordering of ports is given in the specification of M.) The message scheduled by M is delivered to M' and M' is the next active machine. If no such message exists, then the unique master scheduler is the next active machine.

(4) If in step (1) M does not schedule a buffer, then the unique master scheduler is the next active machine.

Under this algorithm, every closed collection of machines induces a unique probability space on the set of runs. In essence, there is no "real" scheduling to be done once a collection is closed, because all scheduling decisions are already specified by machines in the collection.

In the present paper, scheduling for task-PIOAs is *centralized*: it is determined by a global task sequence. Each task determines a unique component and at most one transition in that component. If such a transition is not enabled, then no changes take place.

In our setting, we distinguish between two kinds of nondeterministic choices:

1. High-level choices, such as timing of messages. These are resolved algorithmically by the adversary, which acts as the network.

2. Low-level choices representing inessential ordering of events. In the underlying semantics, these are resolved by a global task sequence.

However, since we have defined a sound notion of simulation relation, these choices can remain unresolved throughout our proofs. That is, it is sufficient to define a simulation relation between two systems, rather than first resolving all nondeterministic choices in one and then trying to mimic the same behavior in the other.

Thus, PIOAs are (purposely) under-specified, so that inessential ordering of events can be abstracted away from our proofs. We think this is a major difference between [PW01, BPW04b] and our work.

A.1.5 Security Properties

In [PW01, BPW04b], the notions of reactive simulatability compare views of a user for complete, closed systems, that is, systems with purely probabilistic behavior.

More precisely, a structure $(\hat{M}_1, S)$ (that is, a set of simple machines with specified service ports available for the user) is said (at least) as secure as another structure $(\hat{M}_2, S)$, iff for every adversary A_1 for $(\hat{M}_1, S)$, there is an adversary A_2 for $(\hat{M}_2, S)$ such that the views of any user H in the configurations $(\hat{M}_1, S, H, A_1)$ and $(\hat{M}_2, S, H, A_2)$ cannot be distinguished. (The quantifiers we use here are those of the *universal simulatability* notion, which is closest to the one we use.) Different indistinguishability notions can be used here, requiring the views to be equal for perfect security, or computationally indistinguishable for computational security for instance.

Even though the security properties we prove in our task-PIOA model are very similar to those described in [PW01, BPW04b], an important difference lies in the fact that our definitions involve comparing task-PIOAs before resolving the non-determinism. As a result, our implementation definitions quantify over all possible task schedules.

A second difference is that, in our computational definition of implementation, we do not compare the views of the user, but the probabilities that the environment (which is the task-PIOA corresponding to the user) outputs a specific *accept* flag. This choice, which sticks more closely to the formulation in the UC framework [Can01], simplified the statement of some properties of our computational implementation notion (see the composition property for instance).

A.1.6 Complexity

In [PW01, BPW04b], a computational realization in terms of interactive probabilistic Turing machines is proposed for the machines presented. A machine is said to be polynomial-time iff it has a realization as a polynomial-time probabilistic Turing machine. Other complexity measures are defined, always using the notion of computational realization.

Our notion of time-bounded task-PIOAs uses a bit-strings encoding of actions, tasks and states, and requires bounds on different probabilistic Turing machines used to decode these actions. It also requires

bounds on the time needed for determining the enabled action in a given task, and for computing the next state from the current state and an action.

Machines have a security parameter k as input, and their computational realization provides a single (uniform) Turing machine used for every value of the security parameter. We conjecture that this notion is equivalent to our notion of uniformly polynomial-time-bounded task-PIOA family.

Our notion of (non-uniform) polynomial-time-bounded task-PIOA family, which is the one we use in practice, is more closely related to the notion of parameterized systems defined in [BPW04a]. The correspondence is not immediate however: systems are sets of structures, which are pairs of machines and service ports.

A.1.7 Proof techniques

In [BPW03], simulatability is proved by using bisimulation with error sets. More precisely, a mapping between states of the two considered systems is defined, and one show that the same input in corresponding states leads to the same output and corresponding states again, except in some specific error cases. Then, it is proved that the probability of these error cases to occur is negligible via reductions on attacks against the underlying cryptographic primitives.

We establish our security proofs in a different way. We use two implementation relations. The first one, $\leq_0$, guarantees that every trace distribution of the first related task-PIOA is also a trace distribution of the second related task-PIOA (without any error probability). We prove this by establishing simulation relations, then by using a soundness result stating that the existence of a simulation relation between two systems guarantees that they are $\leq_0$-related. Rather than relating states, our simulation relation relates probability distributions of execution fragments. Also, we do not prove that probability distributions of execution fragments are related before and after executing any input action, but that, starting from related probability distributions of execution fragments, we reach a countable number of related probability distributions of execution fragments when we execute any task on the first system and a corresponding sequence of tasks in the second system. This type of relation allows us to relate systems with random choices performed at different times, for example. It also allows us to manage the quantifier over all possible task-schedulers in a natural way.

This technique in only used for relating systems in the absence of any computational assumption. Besides this, we define computational assumptions in terms of our approximate implementation relation $\leq_{neg,pt}$ (and prove the equivalence between the standard, computational, formulation of these assumptions and our PIOA version of it). Then, using the composition properties of the $\leq_{neg,pt}$ relation, we prove that the larger task-PIOAs corresponding to the whole protocol model are also $\leq_{neg,pt}$-related. This step does not involve any argument "by contradiction".

A.2 Probabilistic Polynomial-Time Process Calculus

This section is still under construction. It should be available shortly.