Using 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
College of William and Mary
mliskov@cs.wm.edu

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

Olivier Pereira
UCL
olivier.pereira@uclouvain.be

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

December 14, 2005

Abstract

We demonstrate how to carry out cryptographic security analysis of distributed protocols within the Probabilistic I/O Automata framework of Lynch, Segala, and Vaandrager. This framework provides tools for arguing rigorously about the concurrency and scheduling aspects of protocols, and about protocols presented at different levels of abstraction. Consequently, it can help in making cryptographic analysis more precise and less susceptible to errors.

We concentrate on a relatively simple two-party Oblivious Transfer protocol, in the presence of a semi-honest adversary (essentially, an eavesdropper). For the underlying cryptographic notion of security, we use a version of Canetti's Universally Composable security. In spite of the relative simplicity of the example, the exercise is quite nontrivial. It requires taking many fundamental issues into account, including nondeterministic behavior, scheduling, resource-bounded computation, and computational hardness assumptions for cryptographic primitives.

Contents

List of Figures

1 Introduction

Modeling cryptographic protocols and analyzing their security is a tricky business. On the one hand, valid modeling and analysis has to address the concurrency aspects of asynchronous distributed systems, with potentially adversarial scheduling of events. On the other hand, realistic analysis has to accommodate the fact that, in most interesting cases, it is impossible to completely prevent successful attacks against the protocol. Instead, we can only bound the success probability of attacks that use a bounded amount of computational resources. Even worse, given our current state of scientific knowledge, we can typically only make such guarantees based on underlying computational hardness assumptions.

Indeed, cryptographic modeling and analysis is typically complex, involving many subtleties and details, even when the analyzed protocols are simple. Furthermore, analysis is handwritten and often tedious to verify. These factors make security analysis of cryptographic protocols susceptible to errors and omissions. (See, for instance, the errors reported in [S02, HMS03]). They are also obstacles to analyzing more complex cryptographic protocols and systems that use them.

One approach to simplifying cryptographic protocol analysis and improving its correctness is to model cryptographic primitives as "symbolic operations", or "ideal boxes", which represent the security properties of the primitives in an idealized way that involves no error probabilities or computational issues. This approach, first proposed by Dolev and Yao [DY83] and widely used since, indeed simplifies the analysis dramatically. Furthermore, several recent works (e.g., [AR00, BPW03, MW04, CH04]) have demonstrated that this approach can potentially provide cryptographic soundness, in the sense that one can transform secure idealized protocols into secure *concrete* protocols that use concrete cryptographic primitives. This approach is quite promising; however, it does not completely remove the need for cryptographic analysis of protocols. Rather, it only proves security of the overall protocol assuming security of the cryptographic primitives in use. One still has to prove security of these primitives in a full-fledged cryptographic model with all its subtleties. Furthermore, a new abstract model has to be hand-crafted for each new set of cryptographic primitives to be used. While this approach was used successfully for primitives such as encryption and signatures, where a deterministic specification is natural, it is not clear how to extend it to pseudorandom generators or functions

This paper proposes an alternative (in fact, complementary) approach to making cryptographic protocol analysis more mechanical and rigorous, and thus less susceptible to errors. The idea is to *directly* assert the security of a protocol in a concrete model without abstract cryptography, and where security typically holds only for computationally bounded adversaries, and only under computational assumptions. Here the goal is to show that the protocol realizes a specification, where the specification is in itself described as a distributed process, albeit a more abstract and idealized one. Specifically, we propose to express cryptographic protocols, as well as the specification processes, using a variant of the Probabilistic I/O Automata (PIOA) framework developed in the concurrency semantics research community [SL95, LSV03]. Within this framework, we formalize the notion of "implementing a specification" along the lines of the notion of "realizing an ideal functionality " within the universally composable security framework [C01]. We also show how to assert that a cryptographic protocol implements a specification. The rigor and modularity of the underlying PIOA framework allows for analysis that is fully formal and precise, and at the same time understandable and manageable.

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 [S04], as well as methods for mechanizing that process [H05, B05]. 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. Backes, Pfitzman and Waidner [PW00, BPW03], and Canetti [C01] also provide general frameworks for analyzing cryptographic protocols, but they model concurrency quite differently than in the PIOA framework; furthermore, like other cryptographic frameworks based on interactive Turing machines, they are inherently informal as argued above.

Briefly, a PIOA is a kind of abstract automaton. It includes *states*, *start states*, and *actions*, which are classified as input, output, or internal actions. Each action has an associated set of *transitions*, which go from states to probability distributions on states. Thus, PIOAs are capable of expressing random choice. PIOAs can be composed to yield larger PIOAs; in particular, PIOAs modeling 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. In the PIOA framework, 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" component. The framework also includes various kinds of *simulation relations*, which provide sufficient conditions for proving implementation relationships between systems. Like invariants, simulation relations are generally proved by induction.

In all, the PIOA framework allows for a completely rigorous protocol specification and analysis. This stands in contrast to standard cryptographic modeling, where protocols and adversaries are never completely and rigorously specified in terms of the underlying formal model. (For instance, protocols are practically never described in detail in terms of the actual transition function of an interactive Turing machine.)

We provide some high-level motivation for our proposal to use PIOAs for cryptographic protocol analysis. Recall that a typical proof of security of a protocol in a cryptographic model consists of two main parts. The first part consists of describing one or more algorithms for an adversary to perform, typically given access to another adversary. Such an adversary can be either a "simulator" that has to operate in a restricted ("idealized") model, or alternatively, a "reduction," that is, an adversary that performs some assumed-to-be-hard computation. This part of the proof is more "algorithmic" in nature and typically requires some level of human creativity.

The second part of the proof consists of analyzing the adversaries constructed in the first part, and proving some claims regarding their behavior. This part is typically more "mechanical", and boils down to proving that two different probabilistic distributed systems exhibit the same or very similar behaviors. Although the algorithmic part seems relatively hard to mechanize, the analytic part is amenable to mechanization (and eventual automation). However, in typical cryptographic proofs, this analysis is only sketched, and it is here that many errors and oversights occur.

In contrast, precise modeling of asynchronous, probabilistic distributed systems, and proving similarity in behavior of different systems, are among the main strengths of the PIOA framework. Thus, expressing protocols, simulators, and reductions in the PIOA framework, and using the analytical tools from that framework to prove the relevant similarity claims, may take us a long way towards more rigorous, more mechanized, and eventually automated protocol analysis, *while maintaining cryptographic soundness*.

We exemplify this approach by analyzing a relatively simple protocol for a relatively simple task, in a fairly restricted setting. Still, despite its simplicity, this exercise requires dealing with many general issues regarding the modeling of cryptographic analysis within the PIOA framework, including representing resource-bounded computation and scheduling, modeling computational hardness assumptions, representing error probabilities, and resolving several sources of nondeterminism. Overcoming these issues seems to be a prerequisite for performing cryptographic analysis of *any* cryptographic protocol in the PIOA framework. We hope that the modeling and basic formalisms developed here will provide a sound basis for future work in this direction. The next few paragraphs contain a somewhat more detailed sketch of the issues involved and of our modeling approach.

The example. The task we consider is Oblivious Transfer (OT) [R81, EGL85], where a *transmitter* inputs two bits (x_0, x_1), and a *receiver* inputs a selector bit i. The correctness requirement is that the receiver should output x_b. The secrecy requirements are that the receiver should learn *nothing but* x_i and that the transmitter should learn nothing at all. In spite of its apparent simplicity, OT is a very powerful primitive. In fact, it 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 (see, e.g., [GMW87, K89]).

OT is also interesting from an analytical viewpoint, because it imposes secrecy requirements when either party is corrupted, in addition to correctness requirements. (This stands in contrast to the often-analyzed example of key exchange, which imposes no secrecy requirements when either party is corrupted.)

We concentrate on realizing OT in the presence of a *passive* (sometimes called "eavesdropping") adversary, where even corrupted parties continue to follow the protocol instructions. Furthermore, we concentrate on *non-adaptive* corruptions, where the set of corrupted parties is fixed before protocol execution starts. The particular OT protocol we analyze is the classic protocol of [EGL85, GMW87], which uses trap-door permutations (and hard-core predicates for them) as the underlying cryptographic primitive.

The notion of security. We base our definition of cryptographically secure OT (secure against passive, nonadaptive adversaries) on Canetti's definition of OT in the Universally Composable (UC) security framework [C01]. In a nutshell, this definition proceeds as follows: First, an *ideal OT process* is defined—a kind of trusted party that receives inputs from both parties and outputs the correct bit to the receiver. Then a protocol is defined to be a secure OT protocol if it *securely realizes* the OT ideal system, in the sense that for any adversary $\mathcal{A}$ that interacts with the protocol, there exists an adversary ("simulator") $\mathcal{S}$ that interacts with the ideal system, such that no "external environment" $\mathcal{E}$ can tell whether it is interacting with the protocol and $\mathcal{A}$, or alternatively with the ideal process and $\mathcal{S}$.

In our development, we define all the system components—the transmitter and receiver roles in the protocol, the ideal process, the adversaries, and the environment—as PIOAs, and formulate indistinguishability using a definition of implementation for PIOAs.

Modular analysis. The analysis of the protocol is modular, using multiple levels of abstraction in describing the systems of interest. Furthermore, the analysis at each level is broken down into many relatively simple statements that can be proven separately. This enables a treatment that is completely rigorous while being conceptually clear and understandable.

Resolving nondeterminism. In our PIOA models, the various system components make *nondeterministic* as well as probabilistic choices. For example, the order of message delivery by the adversary is left unspecified. Also, we allow nondeterminism in the order in which the different components take steps. We then say that the protocol is secure if the real system "implements" the ideal system, in the sense that *for any* way of resolving the nondeterminism in the real system, *there exists* a way of resolving the nondeterminism in the ideal system, such that the views of the environment $\mathcal{E}$ in the two interactions are the same (or similar). Here we have to make sure that the nondeterministic choices do not give the adversaries effective computational power that is not resource bounded. We do this by essentially restricting the nondeterministic choices to be resolved independently of the values of the inputs and the results of the random choices made during the execution. (Roughly speaking, we say that the nondeterminism is resolved "before the execution starts".)

Resource-bounded adversaries. Capturing resource-bounded adversarial behavior is an essential aspect of cryptographic modeling. One concern, mentioned in the previous paragraph, is to make sure that the method of resolving nondeterministic choices does not give adversaries "back-door access" to "illegitimate computational power". Another concern is to make sure that, after all the nondeterminism is resolved, the operations taken by the adversarial entities in the system are computationally bounded. We guarantee this property by explicitly requiring that all the transitions taken by the schedulers and the adversarial entities in the system are computationally bounded. Specifically, we require that all these transitions are (1) length preserving, in the sense that the description of the end state is no longer than the description of the start state; and (2) computable in probabilistic polynomial time (PPT) in the description of the start state.

Using computational hardness assumptions. To show that the real system "implements" the ideal system one has to consider four cases, depending on which of the two parties are corrupted. When only the transmitter is corrupted, and when both parties are corrupted, it is possible to show that the real system implements the ideal system unconditionally. This allows for relatively straightforward analysis. However, when neither party is corrupted, or when only the receiver is corrupted, implementation can be demonstrated only in a "computational sense", i.e. with respect to PPT adversaries and schedulers. Furthermore, implementation can only be proven assuming the security of the underlying trap-door permutation f. In order to prove such a statement we follow the cryptographic approach of "proof by reduction". That is, given an adversary (or, rather, an "adversarial environment" in our formulation) that breaks the desired implementation relation, construct an adversary that inverts the underlying trapdoor permutation.

We take a slightly different approach: we first formulate the security property of the trap-door permutation f in terms of an implementation relation on PIOAs. That is, we formulate a "concrete TDP" PIOA and an "abstract TDP" PIOA, and then show that if f is a trap-door one-way permutation then the concrete TDP PIOA implements the abstract TDP PIOA. Then, the rest of the analysis is performed assuming that the concrete TDP PIOA implements that abstract TDP PIOA. This allows us to perform the entire analysis in the PIOA framework using the implementation relation and without explicit proofs by reduction.

We remark that the actual analysis involves a few more steps than what is indicated in the above sketch. First, instead of using the security of f directly, we use the security of a hard-core predicate $B()$ for f. (Recall that any one way function, trap-door permutations being a special case, has a hard-core predicate [GL89].) That is, we use the fact that if f is chosen uniformly from a family of one-way trap-door permutations, x is chosen uniformly from the domain of f, and b is a uniformly chosen bit, then the triple $(f, f(x), B(x))$ is polynomial-time indistinguishable from the triple $(f, f(x), b)$. Furthermore, we use the fact that seeing two hard-core bits of two pre-images of randomly chosen values is still indistinguishable from seeing two random bits.

Extending the PIOA framework. Following the usual proof methods for distributed algorithms, we have decomposed our proofs into several stages, with general transitivity results used to combine the results of the stages. 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 each proof.

Producing this proof required us to develop two new kinds of theory: First, we extended traditional PIOA theory in two ways:

- We defined a new notion of *tasks*, which provide a mechanism to resolve nondeterministic choices.
- We defined a new kind of *simulation relation*, which allows one to establish a correspondence between probability distributions on states at two levels of abstraction, and which allows splitting of distributions in order to show that individual steps preserve the correspondence.

Second, we developed a new theory for time-bounded PIOAs, specifically:

- We defined *time-bounded PIOAs*, which impose time bounds on the individual steps of the PIOAs.
- We defined a new *approximate, time-bounded, implementation relationship* between time-bounded PIOAs, which is sufficient to capture the typical relationships between cryptographic primitives and the abstractions they are supposed to implement.

In the multi-stage proofs, most of the stages represent exact (not approximate) implementations; we prove all these using standard PIOA theory, extended with our new simulation relation. The techniques for showing this are fairly standard in the distributed algorithms research literature, based on proving invariants and simulation relationships by induction on the number of steps in an execution. The remaining stages involve replacement of a cryptographic primitive with a random counterpart; we prove that these satisfy our approximate implementation relationship. The techniques for showing this are based on recasting the definitions of the cryptographic primitives in terms of approximate implementation relationships, and then combining these primitives with other components in various ways that preserve the implementation relationships. Transitivity results allow us to combine all the implementation relationships proved at all the stages to obtain an overall approximate implementation relationship between the Oblivious Transfer algorithm and its property specification.

2 Informal Description

We consider an oblivious transfer protocol in which 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. The following is an informal description of the desired behavior:

Oblivious Transfer Functionality $\mathcal{F}$
On inputs (x_0, x_1) from T, record (x_0, x_1) On input i from R, send x_i to R

We analyze the following protocol for realizing this functionality. The protocol was first proposed in [GMW87].

Oblivious Transfer Protocol
On inputs (x_0, x_1) for T and i for R. T selects a random trap-door permutation $f : D \to D$ 1. $T \to R$: f R selects two random elements $y_0, y_1 \in D$ 2. $R \to T$: $(f^{1-i}(y_0), f^i(y_1))$ T receives these values as (z_0, z_1) 3. $T \to R$: $(B(f^{-1}(z_0)) \oplus x_0, B(f^{-1}(z_1)) \oplus x_1)$ where B is a hard-core predicate for f. R receives these values as (b_0, b_1). Finally, R outputs $B(y_i) \oplus b_i$.

At a very high level, the analysis proceeds as follows. We define two systems, the "real system", which captures the protocol execution, and the "ideal system" which captures the ideal specification for OT. Showing that the protocol is correct and secure amounts to showing that the real system "implements" the ideal system, in a certain sense.

In the real system, we consider an adversary $\mathcal{A}$ interacting with the two parties $\mathcal{T}$ and $\mathcal{R}$ executing the protocol. All communications between $\mathcal{T}$ and $\mathcal{R}$ are mediated by the adversary $\mathcal{A}$. An environment $\mathcal{E}$ supplies inputs and receives outputs to/from $\mathcal{T}$ and $\mathcal{R}$, and also interacts with $\mathcal{A}$. In the security literature, all the parties are usually described as Interacting Turing Machines (ITMs), which interact by sharing input and output tapes. The adversary is activated first, and can write on the input tape of one other ITM. Then, when it stops, the ITM which had its input tape written on is activated, and so on.

Besides deciding how the messages are transmitted, the adversary $\mathcal{A}$ can decide to *corrupt* a party, in which case he gains access to the inputs of that party. In this paper, we restrict attention to the case of a *semi-honest* adversary, which means that the parties continue to follow the protocol definition even after being corrupted. Furthermore, we will assume that the adversary is *static*, in the sense that it decides which parties to corrupt before the beginning of the protocol execution.

In the ideal system, we consider a *simulator* $\mathcal{S}$ interacting with an *ideal functionality* $\mathcal{F}$, which is an incorruptible trusted party that is assumed to perform the protocol task. The simulator $\mathcal{S}$ and the functionality $\mathcal{F}$ also interact with the same environment $\mathcal{E}$ as in the real system. The simulator $\mathcal{S}$ has access to the inputs and outputs of the corrupted parties.

We say that the protocol consisting of $\mathcal{T}$, $\mathcal{R}$ *securely realizes* the functionality $\mathcal{F}$ if, for any adversary $\mathcal{A}$ and any environment $\mathcal{E}$, there exists a simulator $\mathcal{S}$ such that the real system consisting of $\mathcal{T}$, $\mathcal{R}$, $\mathcal{A}$ and $\mathcal{E}$ "looks like" the ideal system consisting of $\mathcal{F}$, $\mathcal{S}$, and $\mathcal{E}$, from the point of view of the environment $\mathcal{E}$.

In showing that such a real system looks like a corresponding ideal system, the simulator is generally constructed in terms of variants of the adversary, transmitter, and receiver in the real system.

In the rest of this paper, we develop these ideas formally, in terms of Probabilistic I/O Automata.

3 Mathematical Foundations

This section contains mathematical foundations for the rest of the paper, starting in Section 3.1 with preliminary definitions for sets, functions, and probability measures. Then, in Section 3.2, we review definitions and results for PIOAs. We introduce our new "task" mechanism for resolving nondeterminism in PIOAs in Section 3.3, which leads to a definition of task-PIOAs. Section 3.4 introduces time-bounded task-PIOAs, that is, task-PIOAs whose computation time is bounded by particular functions. Finally, Section 3.5 introduces families of time-bounded task-PIOAs, with polynomial-time task-PIOAs as a special case.

3.1 Preliminaries

3.1.1 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.

3.1.2 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 kinds of simulation relation for task-PIOAs, in Section 3.3.8. 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 $\mu : \mathcal{F} \rightarrow [0, \infty]$ 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 3.1 *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 3.2 *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 3.3 *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 $\mathcal{E}(R)$. $\square$

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 3.52; there, the two functions f and g apply corresponding sequences of tasks to corresponding measures on states.

Lemma 3.4 *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 3.2, 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.

$\square$

3.2 Probabilistic I/O Automata

This section contains standard definitions for PIOAs, extracted from the prior literature—see, e.g., [SL95, LSV03]. After presenting the basic definitions of PIOAs and their executions, in Section 3.2.1, we give careful definitions for the σ-field of execution fragments and the σ-field of traces of a PIOA, in Section 3.2.2 In terms of these σ-fields, we give careful definitions (and some basic results) for probabilistic executions and trace distributions, in Section 3.2.3. The remaining two subsections define the composition and hiding operations for PIOAs.

3.2.1 Basic definitions

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 3.5 *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 $trace(\alpha) = \beta$.

3.2.2 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 3.6 *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 3.7 *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 3.8 *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 3.9 *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 3.10 *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 3.11 *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 3.12 *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 3.13 *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 3.14 *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*.

3.2.3 Probabilistic executions and trace distributions

Having established some groundwork in Section 3.2.2, 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 3.15 *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 3.16 *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:*

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

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 3.17 *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 3.18 *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 3.17 after observing that $\epsilon_{\sigma,\alpha}(C_\alpha) = 1$. □

Lemma 3.19 *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 3.17, 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 3.18 (with α instantiated as $\tilde{\alpha}$). □

Lemma 3.20 *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*

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

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 $\alpha' \leq \alpha$, $\epsilon_{\sigma,\alpha'}(\alpha) = \epsilon_{\sigma,\alpha'}(C_\alpha)\sigma(\alpha)(\perp)$, we derive $\epsilon_{\sigma,\mu}(\alpha) = \sum_{\alpha'\leq\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha)\sigma(\alpha)(\perp)$. Observe that $\sigma(\alpha)(\perp)$ is a constant with respect to α', and thus can be moved out of the sum, yielding $\epsilon_{\sigma,\mu}(\alpha) = (\sum_{\alpha'\leq\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha))(\sigma(\alpha)(\perp))$.

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 3.18. □

Lemma 3.21 *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 3.19, 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 3.22 *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 3.21, 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 3.17, $\epsilon_{\sigma,\mu}(C_q) = \mu(C_q)$, 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 $\alpha = \tilde{\alpha} a q$. By Lemma 3.19,

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

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,

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

Also by Lemma 3.19, we obtain that

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

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 $tdists(\mathcal{P})$.

3.2.4 Composition

We define composition for PIOAs:

Definition 3.23 *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.

3.2.5 Hiding

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

Definition 3.24 *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 $H_{\mathcal{P}} = H_{\mathcal{P}} \cup S$.*

3.3 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. The state equivalence relation helps us to express certain technical restrictions on the transitions. This aggregation will be used to weaken the power of the scheduler, by forcing it to ignore differences such as results of secret random choices.

We begin by defining task-PIOAs, in Section 3.3.1. Then we define task schedulers, in Section 3.3.2, which are a variant of our schedulers with coarser granularity (they schedule tasks rather than specific transitions). Section 3.3.3 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 3.3.6, we define the notion of an "environment" for a task-PIOA. We use this, in Section 3.3.7, to define what it means for one task-PIOA to implement another. Finally, in Section 3.3.8, we define our new kind of simulation relation between closed task-PIOAs, and prove that it is sound with respect to our implementation notion.

3.3.1 Task-PIOAs

Definition 3.25 *We define a* task-PIOA*, to be a triple* $\mathcal{T} = (\mathcal{P}, RA, RS)$*, where*

- $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$ *is a PIOA (satisfying next-transition determinism).*
- RA *is an equivalence relation on the action set* A*.*
  We refer to the equivalence classes of RA *as* tasks*. We require that* RA *respect action types: each* $T \in RA$ *is a subset of* I*,* O*, or* H*. We refer to the tasks as* input tasks, output tasks*, or* internal tasks*, respectively.*
- RS *is an equivalence relation on the state set* Q*.*

A task T *is* enabled in *a state* q *if there is some action 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 (rather strong) conditions:

1. **Next-action determinism:** *For every state* $q \in Q$ *and every output or internal task* $T \in RA$*, there is at most one action* $a \in T$ *that is enabled in* q
2. **Random-choice consistency:** *If* $(q, a, \mu) \in D$*, then* $supp(\mu) \subseteq S$ *for some* $S \in RS$*.*
3. **Transition consistency:** *Suppose that* $q_1 \equiv_{RS} q_2$*,* $a_1 \equiv_{RA} a_2$*,* $\{(q_1, a_1, \mu_1), (q_2, a_2, \mu_2)\} \subseteq D$*, then* $supp(\mu_1) \cup supp(\mu_2) \subseteq S$ *for some* $S \in RS$*.*
4. **Enabling consistency:** *If* $q_1 \equiv_{RS} q_2$*,* $a_1 \in O \cup H$*, and* $(q_1, a_1, \mu_1) \in D$*, then there exists a transition* $(q_2, a_2, \mu_2) \in D$ *such that* $a_1 \equiv_{RA} a_2$*.*

We denote the relations RA and RS of a task-PIOA $\mathcal{T}$ by $RA_{\mathcal{T}}$ and $RS_{\mathcal{T}}$. If S is a set of states of $\mathcal{P}$ such that all states in S are RS-equivalent, then we write $[S]_{\mathcal{T}}$ to denote the unique equivalence class $S' \in RS$ such that $S \subseteq S'$. Similarly, if μ is a discrete distribution on states of $\mathcal{P}$ such that all states in $supp(\mu)$ are RS-equivalent, then we write $[\mu]_{\mathcal{T}}$ to denote the unique equivalence class $S' \in RS$ such that $supp(\mu) \subseteq S'$. We drop the subscript $\mathcal{T}$ when we think it should be clear from the context.

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

3.3.2 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.

In this paper, our notion of task scheduler is *oblivious*—that is, it is just a sequence of tasks. In the security protocol setting, we would like also to consider task schedulers that can depend on partial information about the past execution, in particular, on the portion of the execution that is visible to the adversarial components. However, this extension will require significant generalizations to the machinery we have developed so far, and we leave it for future work.

Definition 3.26 *Let* $\mathcal{T} = (\mathcal{P}, RA, RS)$ *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* RA*.*

3.3.3 Probabilistic executions and trace distributions

We next describe how a given task scheduler generates a generalized probabilistic execution fragment given a starting measure μ on finite execution fragments. We do this by defining a function $apply(,)$ that takes a discrete measure μ on finite execution fragments and a task scheduler ρ and returns the result of applying ρ from μ, which is a measure on execution fragments. We define $apply(,)$ first for the empty sequence of tasks, then for a single task, then for a finite sequence of tasks, and finally for an infinite sequence of tasks.

Definition 3.27 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ be a closed task-PIOA where $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$. Then apply(,) is a function that takes a discrete probability measure on finite execution fragments and a task scheduler 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. *If T is a single task, then for each finite execution fragment α, $apply(\mu, T)(\alpha) = p_1(\alpha) + p_2(\alpha)$, where:*

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

   Next-transition determinism implies that there can be only one such transition, 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 a finite sequence of tasks $\rho' T$, then $apply(\mu, \rho) = apply(apply(\mu, \rho'), T)$.*
4. *If ρ is an infinite sequence, then let ρ_i be the prefix of ρ consisting of the first i tasks of ρ, and let ϵ_i be $apply(\mu, \rho_i)$. Then $apply(\mu, \rho) = \lim_{i \to \infty}(\epsilon_i)$. The limit is well-defined due to Lemma 3.35.*

Lemma 3.28 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ be a closed task-PIOA, and let ρ be a finite task scheduler. Let q_1 and q_2 be two states of $\mathcal{P}$ such that $q_1 \equiv_{RS} q_2$. Let μ_1 and μ_2 be $apply(q_1, \rho)$ and $apply(q_2, \rho)$, respectively.*
Then $supp(lstate(\mu_1)) \cup supp(lstate(\mu_2)) \subseteq S$, for some $S \in RS$.

Proof. By induction on the length of ρ, using the enabling-consistency and transition-consistency properties for task-PIOAs. ☐

The next proposition states that $apply(\cdot, \rho)$ distributes over convex combinations of probability measures. We start with a few preliminary lemmas.

Lemma 3.29 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ be a closed 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, $\mu(\alpha) = p_2(\alpha) + \sum_{(a,q):\alpha a q \in frags^*(\mathcal{P})} p_1(\alpha a q)$.*

Proof. For Part 1, the fact that $p_1(q) = 0$ for each state q follows trivially by definition of $p_1(q)$.

For Part 2, consider a finite execution fragment α. We observe the following facts:

1. 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 $\alpha a q$ is an execution fragment, we claim that $p_1(\alpha a q) = 0$: 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.
2. If T is enabled from $lstate(\alpha)$, then, trivially, $p_2(\alpha) = 0$. Furthermore, we claim that $\mu(\alpha) = \sum_{(a,q)} p_1(\alpha a q)$: Indeed, there exists only one action $b \in T$ that is enabled from $lstate(\alpha)$. By definition of p_1, $p_1(\alpha a q) = 0$ if $a \neq b$ (either $a \notin T$ or a is not enabled from $lstate(\alpha)$). Thus, $\sum_{(a,q)} p_1(\alpha a q) = \sum_q p_1(\alpha b q) = \sum_q \mu(\alpha)\mu_{\alpha,b}(q)$. This in turn is equal to $\mu(\alpha)$ since $\sum_q \mu_{\alpha,b}(q) = 1$.

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

Lemma 3.30 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ be a closed 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 showing 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 the definition of $apply(,)$. To show that ϵ' is in fact a probability measure, we show that $\sum_{\alpha \in frags^*(\mathcal{P})} \epsilon'(\alpha) = 1$. By Item 2 of the definition of $apply(,)$, $\sum_{\alpha \in frags^*(\mathcal{P})} \epsilon'(\alpha) = \sum_{\alpha \in frags^*(\mathcal{P})} (p_1(\alpha) + p_2(\alpha))$. By rearranging terms, $\sum_{\alpha \in frags^*(\mathcal{P})} \epsilon'(\alpha) = \sum_q p_1(q) + \sum_{\alpha \in frags^*(\mathcal{P})} (p_2(\alpha) + \sum_{(a,q):\alpha a q \in frags^*(\mathcal{P})} p_1(\alpha a q))$. By Lemma 3.29, the right side becomes $\sum_{\alpha \in frags^*(\mathcal{P})} \epsilon(\alpha)$. Since $\sum_{\alpha \in frags^*(\mathcal{P})} \epsilon(\alpha) = 1$, then also $\sum_{\alpha \in frags^*(\mathcal{P})} \epsilon'(\alpha) = 1$, as needed. □

Lemma 3.31 *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 3.27, $p_1(\alpha) = (\sum_i p_i \mu_i)(\alpha')\rho(q)$, 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 3.27, $p_1(\alpha) = 0$, 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 3.27, $p_2(\alpha) = (\sum_i p_i \mu_i)(\alpha)$, 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 3.27, $p_2(\alpha) = 0$, and, for each i, $p_2^i(\alpha) = 0$. Thus, $p_2(\alpha) = \sum_i p_i p_2^i(\alpha)$ trivially. □

Proposition 3.32 *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 3.27, $apply(\sum_i p_i \mu_i, \rho' T) = apply(apply(\sum_i p_i \mu_i, \rho'), T)$. 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 3.30, for each i, $apply(\mu_i, \rho')$ is a discrete probability measure over finite execution fragments. By Lemma 3.31, $apply(\sum_i p_i \, apply(\mu_i, \rho'), T) = \sum_i p_i \, apply(apply(\mu_i, \rho'), T)$, and by Definition 3.27, 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. □

We now prove that $apply(\mu, \rho)$ returns a generalized probabilistic execution fragment generated by μ (and some ordinary scheduler). This result is stated as Proposition 3.41. Our proof uses a series of auxiliary lemmas.

Lemma 3.33 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ be a closed task-PIOA. Let $\mu' = apply(\mu, T)$. 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 $\alpha = \tilde{\alpha} a q$ and $a \notin T$, then $\mu'(C_\alpha) = \mu(C_\alpha)$.*

3. *If $\alpha = \tilde{\alpha} a q$ 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 3.29, $\mu(C_\alpha) = \sum_{\alpha' | \alpha \leq \alpha'} (p_2(\alpha') + \sum_{(a,q):\alpha' a q \in frags^*(\mathcal{P})} p_1(\alpha' a q)) = \sum_{\alpha' | \alpha \leq \alpha'} (p_1(\alpha') + p_2(\alpha')) - p_1(\alpha)$. 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 3.29, yielding $\mu'(C_\alpha) = \mu(C_\alpha)$. If $\alpha = \tilde{\alpha} a q$ and $a \notin T$, then $p_1(\alpha) = 0$ by definition, yielding $\mu'(C_\alpha) = \mu(C_\alpha)$. Finally, if $\alpha = \tilde{\alpha} a q$ 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 3.34 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ be a closed 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 3.33. □

Lemma 3.35 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ be a closed 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 3.34 for the inductive step. □

Lemma 3.36 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ 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 by the definitions by defining a scheduler σ such that $\sigma(\alpha)(tr) = 0$ for each finite execution fragment α and each transition tr. □

Lemma 3.37 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ be a closed 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. Then the result for infinite ρ's follows by limit. 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 of $apply(,)$, $apply(\mu, \rho) = apply(\epsilon, T)$. By induction, $\epsilon(C_q) = \mu(C_q)$. We show that $apply(\epsilon, T)(C_q) = \epsilon(C_q)$, which suffices.

Let ϵ' be $apply(\epsilon, T)$. By definition of cone, $\epsilon'(C_q) = \sum_{\alpha : q \leq \alpha} \epsilon'(\alpha)$. Since, by Lemma 3.30, both ϵ and ϵ' are measures over finite execution fragments, 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 the definition of $apply(\epsilon, T)$. Then $\epsilon'(C_q) = \sum_{\alpha \in execs^*(\mathcal{P}) : q \leq \alpha} (p_1(\alpha) + p_2(\alpha))$. By rearranging terms, we get $\epsilon'(C_q) = p_1(q) + \sum_{\alpha \in execs^*(\mathcal{P}) : q \leq \alpha} (p_2(\alpha) + \sum_{(a,s)} p_1(C_{\alpha a s}))$. By Lemma 3.29, the right side of the equation above is $\sum_{\alpha : q \leq \alpha} \epsilon(\alpha)$, that is, $\epsilon(C_q)$, as needed. □

Lemma 3.38 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ be a closed 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. Let σ be a scheduler that, together with μ, generates ϵ (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 $\sigma'(\alpha)(tr) = 0$, otherwise,

$$\sigma'(\alpha)(tr) = \begin{cases} \dfrac{\epsilon(C_\alpha) - \mu(C_\alpha - \{\alpha\})}{\epsilon'(C_\alpha) - \mu(C_\alpha - \{\alpha\})}(\sigma(\alpha)(tr) + \sigma(\alpha)(\perp)) & \text{if } tr \in D(lstate(\alpha)) \text{ and } act(tr) \in T, \\ \dfrac{\epsilon(C_\alpha) - \mu(C_\alpha - \{\alpha\})}{\epsilon'(C_\alpha) - \mu(C_\alpha - \{\alpha\})}\sigma(\alpha)(tr) & \text{otherwise} \end{cases}$$

where $D(lstate(\alpha))$ denotes the set of transitions of D with source state $lstate(\alpha)$ and $act(tr)$ denotes the action that occurs in tr. 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 3.17, $\epsilon_{\sigma,\mu}(C_q) = \mu(C_q)$ and $\epsilon_{\sigma',\mu}(C_q) = \mu(C_q)$. Thus, $\epsilon_{\sigma',\mu}(C_q) = \epsilon_{\sigma,\mu}(C_q)$. The right-hand-side is in turn equal to $\epsilon(C_q)$ by definition, which is equal to $\epsilon'(C_q)$ by Lemma 3.37. Thus, $\epsilon_{\sigma',\mu}(C_q) = \epsilon'(C_q)$, as needed.

For the inductive step, let $\alpha = \tilde{\alpha}aq$. By Lemma 3.17 and Equation (1), the definition of the measure of a cone, 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, $tr_{\tilde{\alpha},a}$ and $\mu_{\tilde{\alpha},a}$ are defined. By expanding $\mu_{\sigma'(\tilde{\alpha})}(a, q)$ in the equation above, we get

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

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 3.19, $\epsilon_{\sigma',\mu}(C_\alpha) = \mu(C_\alpha)$. We show that $\epsilon'(C_\alpha) = \mu(C_\alpha)$, which suffices.

   By Lemma 3.34, $\epsilon(C_{\tilde{\alpha}}) \leq \epsilon'(C_{\tilde{\alpha}})$. Thus, combining with $\epsilon'(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) = 0$, we get $\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) \leq 0$. On the other hand, from Lemma 3.18, and from $\epsilon = \epsilon_{\sigma,\mu}$, we derive $\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) \geq 0$. Thus, $\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) = 0$.

   By Lemma 3.19, since $\epsilon_{\sigma,\mu} = \epsilon$ and $\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) = 0$, we get $\epsilon(C_\alpha) = \mu(C_\alpha)$.

   By Lemma 3.34, since $C_{\tilde{\alpha}} - \{\tilde{\alpha}\}$ is a union of cones, $\mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) \leq \epsilon(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$. By 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 $\epsilon(C_{\tilde{\alpha}}) = \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$, from the previous inequalities we derive $\epsilon(C_{\tilde{\alpha}}) + \epsilon(\{\tilde{\alpha}\}) \leq \epsilon(C_{\tilde{\alpha}})$, which implies $\epsilon(\{\tilde{\alpha}\}) = 0$. By Lemma 3.33, cases 2 and 3, $\epsilon'(C_\alpha) = \epsilon(C_\alpha)$, which is equal to $\mu(C_\alpha)$, as needed.

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

   By Equation (2) and by definition of σ',

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

   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 3.18 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, by 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

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

   Since, by definition, $\epsilon = \epsilon_{\sigma,\mu}$, by Lemma 3.19, the right side of the equation above is $\epsilon(C_\alpha)$. By Lemma 3.33, Part 2, $\epsilon(C_\alpha) = \epsilon'(C_\alpha)$. Thus, $\epsilon_{\sigma',\mu}(C_\alpha) = \epsilon'(C_\alpha)$, as needed.

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

   By following the same approach as in the previous case,

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

   Since, as shown in the previous case, $\epsilon(C_\alpha) = \mu(C_\alpha) + (\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\,\sigma(\tilde{\alpha})(tr_{\tilde{alpha},a})\mu_{\tilde{\alpha},a}(q)$, the equation above becomes

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

   By replacing $(\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\sigma(\tilde{\alpha})(\perp)$ according to Lemma 3.20, and observing that, by definition, $\epsilon = \epsilon_{\sigma,\mu}$, we get

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

   Then, the result follows by Lemma 3.33, Part 3.

□

Lemma 3.39 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ be a closed 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 3.36 for the base case and Lemma 3.38 for the inductive step. □

Lemma 3.40 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ be a closed 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 be the prefix of ρ consisting of the first i tasks of ρ, and let ϵ_i be $apply(\mu, \rho_i)$. By Lemmas 3.39 and 3.35 $\epsilon_0, \epsilon_1, \ldots$ is a chain of generalized probabilistic execution fragments generated by μ. By Proposition 3.22, $\lim_{i\to\infty} \epsilon_i$ is a generalized probabilistic execution fragment generated by μ, which suffices since, by definition, $apply(\mu, \rho)$ is $\lim_{i\to\infty} \epsilon_i$. □

Now we can prove Proposition 3.41, our main target. It says that any probability measure on execution fragments that is generated by $apply(\mu, \rho)$ for any μ and ρ, is a "standard" probability measure on execution fragments—one that is generable from μ using a traditional scheduler.

Proposition 3.41 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ be a closed 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 3.39 and 3.40. □

Lemma 3.42 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ be a closed 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 by Lemma 3.34. For the limit property, if the sequence $\rho_1, \rho_2, \ldots$ is finite, then the result is immediate. Otherwise, it is enough to observe that the sequence $\epsilon_1, \epsilon_2, \ldots$ is a sub-sequence of the sequence used in the definition of $apply(\mu, \rho_1\rho_2\ldots)$ that has 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 $\{tdist(\rho) : \rho \text{ is a task scheduler for } \mathcal{T}\}$.

3.3.4 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 3.43 *To define composition of task-PIOAs, we need an additional compatibility requirement. Namely, we say that two task-PIOAs $\mathcal{T}_1 = (\mathcal{P}_1, RA_1, RS_1)$ and $\mathcal{T}_2 = (\mathcal{P}_2, RA_2, RS_2)$ are* compatible *provided that the following conditions are satisfied:*

1. *The underlying automata $\mathcal{P}_1$ and $\mathcal{P}_2$ are compatible.*
2. *For every task T_1 of $\mathcal{T}_1$ and every task T_2 of $\mathcal{T}_2$, either $T_1 = T_2$ or $T_1 \cap T_2 = \emptyset$.*

Then we define the composition $\mathcal{T} = (\mathcal{P}, RA, RS)$ *of two compatible task-PIOAs $\mathcal{T}_1 = (\mathcal{P}_1, RA_1, RS_1)$ and $\mathcal{T}_2 = (\mathcal{P}_2, RA_2, RS_2)$, denoted by $\mathcal{T}_1 \| \mathcal{T}_2$, as follows:*

- *$\mathcal{P} = \mathcal{P}_1 \| \mathcal{P}_2$.*
- *RA is $RA_1 \cup RA_2$.*
- *RS is the equivalence relation defined as follows: $q \equiv_{RS} q'$ iff $q \lceil \mathcal{P}_i \equiv_{RS_i} q' \lceil \mathcal{P}_i$ for every $i \in \{1, 2\}$.*

We sometimes write $q \lceil \mathcal{T}_i$ to denote $q \lceil \mathcal{P}_i$ for $i \in \{0, 1\}$.

Proposition 3.44 *$\mathcal{T}_1 \| \mathcal{T}_2$ is a task-PIOA.*

Proof. We must show that $\mathcal{T}_1 \| \mathcal{T}_2$ satifies the consistency properties 1–4 in the definition of a task-PIOA.

1. *Next-action determinism:* Let (q_1, q_2) be a state of $\mathcal{P}_1 \| \mathcal{P}_2$ and T an output or internal task in RA. Then T is an output or internal task of one of the two components, without loss of generality, of $\mathcal{P}_1$. By next-action determinism of $\mathcal{T}_1$, at most one action $a \in T$ is enabled in q_1, and hence at most that same action a is enabled in (q_1, q_2).

2. *Random-choice consistency:* Let $((q_1, q_2), a, \mu_1 \times \mu_2)$ be a transition of $\mathcal{P}_1 \| \mathcal{P}_2$. Consider each $i = 1, 2$: If a is an action of $\mathcal{P}_i$, then (q_i, a, μ_i) is a transition of $\mathcal{P}_i$. Then by random-choice consistency of $\mathcal{T}_i$, $supp(\mu_i) \subseteq S_i$ for some $S_i \in RS_i$. On the other hand, if a is not an action of $\mathcal{P}_i$, then $supp(\mu_i)$ is just $\{q_i\}$, which trivially is a subset of S_i for some $S_i \in RS_i$.

   Now, $supp(\mu_1 \times \mu_2) = supp(\mu_1) \times supp(\mu_2) \subseteq S_1 \times S_2$. By definition of RS in terms of RS_1 and RS_2, $S_1 \times S_2 \in RS$, so this yields the conclusion.

3. *Transition consistency:* Suppose that $((q_1^1, q_2^1), a^1, \mu_1^1 \times \mu_2^1)$ and $((q_1^2, q_2^2), a^2, \mu_1^2 \times \mu_2^2)$ are two transitions of $\mathcal{P}_1 \times \mathcal{P}_2$ and suppose that $(q_1^1, q_2^1) \equiv_{RS} (q_1^2, q_2^2)$ and $a^1 \equiv_{RA} a^2$. Then $q_1^1 \equiv_{RS_1} q_1^2$ and $q_2^1 \equiv_{RS_2} q_2^2$.

   Consider each $i = 1, 2$: If a^1 is an action of $\mathcal{P}_i$, then since $a^1 \equiv_{RA} a^2$, and by definition of the tasks of $\mathcal{T}_1 \| \mathcal{T}_2$ and compatibility, a^2 is also an action of $\mathcal{P}_i$. Then (q_i^1, a^1, μ_i^1) and (q_i^2, a^2, μ_i^2) are both transitions of $\mathcal{P}_i$. Then by transition consistency of $\mathcal{P}_i$, $supp(\mu_i^1) \cup supp(\mu_i^2) \subseteq S_i$ for some $S_i \in RS_i$.

On the other hand, if a_1 is not an action of $\mathcal{P}_i$, then neither is a^2. In this case, $supp(\mu_i^1)$ is just q_i^1 and $supp(\mu_i^2)$ is q_i^2; since $q_i^1 \equiv_{RS_i} q_i^2$, there again exists a single equivalence class $S_i \in RS_i$ such that $supp(\mu_i^1) \cup supp(\mu_i^2) \subseteq S_i$.

Now, $supp(\mu_1^1 \times \mu_2^1) = supp(\mu_1^1) \times supp(\mu_2^1) \subseteq S_1 \times S_2$, and similarly $supp(\mu_1^2 \times \mu_2^2) = supp(\mu_1^2) \times supp(\mu_2^2) \subseteq S_1 \times S_2$, So, $supp(\mu_1^1 \times \mu_2^1) \cup supp(\mu_1^2 \times \mu_2^2) \subseteq S_1 \times S_2$. Since $S_1 \times S_2 \in RS$, this yields the conclusion.

4. *Enabling consistency:* Suppose that $(q_1^1, q_2^1) \equiv_{RS} (q_1^2, q_2^2)$, a^1 is an output or internal action of $\mathcal{P}_1 \| \mathcal{P}_2$, and $((q_1^1, q_2^1), a^1, \mu_1^1 \times \mu_2^1)$ is a transition of $\mathcal{P}_1 \| \mathcal{P}_2$. Then a^1 is an output or internal action of one of the two component automata, without loss of generality, of $\mathcal{P}_1$. Then $q_1^1 \equiv_{RS_1} q_1^2$ and (q_1^1, a^1, μ_1^1) is a transition of $\mathcal{P}_1$ Then by enabling consistency for $\mathcal{T}_1$, there exists a transition (q_1^2, a^2, μ_1^2) of $\mathcal{P}_1$ such that $a^1 \equiv_{RA} a^2$.

   Now, if a^2 is an action of $\mathcal{P}_2$, then it must be an input, and so is enabled from all states. Therefore, in this case, there exists a transition (q_2^2, a^2, μ_2^2) of $\mathcal{P}_1$. Then $((q_1^2, q_2^2), a^2, \mu_1^2 \times \mu_2^2)$ is a transition of $\mathcal{P}_1 \| \mathcal{P}_2$, as needed.

   On the other hand, if a^2 is not an action of $\mathcal{P}_2$, then $((q_1^2, q_2^2), a^2, \mu_1^2 \times \mu_2^2)$ is a transition of $\mathcal{P}_1 \| \mathcal{P}_2$, where μ_2^2 is the Dirac distribution $\delta(q_2^2)$. Either way, we have the needed transition.

□

3.3.5 Hiding

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

Definition 3.45 *Let $\mathcal{T} = (\mathcal{P}, RA, RS)$ be a task-PIOA where $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$, and let $\mathcal{U} \subseteq RA$ 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 $(hide(\mathcal{P}, S), RA, RS)$.*

3.3.6 Environments

We define the notion of environment as follows.

Definition 3.46 *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.

The following lemma about the existence of environments will be useful later, for example, in the proof of Lemma 3.62, which asserts the transitivity property of our time-bounded implementation notion. It says that, if a set of task PIOAs $\{\mathcal{T}_1, \cdots, \mathcal{T}_n, \mathcal{U}\}$ are comparable task-PIOAs (see Definition 3.48) and $\mathcal{E}$ is an environment for every $\mathcal{T}_i$, then $\mathcal{E}$ can be transformed into an environment $\mathcal{E}'$ for both every $\mathcal{T}_i$ and $\mathcal{U}$ through some renaming of the actions of $\mathcal{E}$ which does not modify its way of interacting with the $\mathcal{T}_i$'s.

Lemma 3.47 *Suppose $\mathcal{T}_1, \ldots, \mathcal{T}_n$, $\mathcal{U}$ and $\mathcal{E}$ are task-PIOAs, all $\mathcal{T}_i$'s and $\mathcal{U}$ are comparable, and $\mathcal{E}$ is an environment for every $\mathcal{T}_i$. Then there exists a task-PIOA $\mathcal{E}'$ that is isomorphic to $\mathcal{E}$, such that $\mathcal{E}'$ is an environment for every $\mathcal{T}_i$ and $\mathcal{U}$; this isomorphism preserves all external actions of $\mathcal{E}$.*

Proof. We define an environment $\mathcal{E}'$ by using a bijective renaming function $f : A_{\mathcal{E}} \to A_{\mathcal{E}'}$ that preserves the partition into input, output and internal actions. For simplicity, we assume that, if $f(a) \neq a$, then $f(a) \notin \bigcup_{i=1\ldots n} A_{\mathcal{T}_i} \cup A_{\mathcal{U}} \cup A_{\mathcal{E}}$. We also write $I_{\mathcal{T}}$ and $O_{\mathcal{T}}$ instead of $I_{\mathcal{T}_i}$ and $O_{\mathcal{T}_i}$, as all $\mathcal{T}_i$ have the same inputs and outputs.

The isomorphism condition implies that the renaming function f is the identity for all actions in $(I_{\mathcal{E}} \cap O_{\mathcal{T}}) \cup (O_{\mathcal{E}} \cup I_{\mathcal{T}})$: it does not alter the communication between the $\mathcal{T}_i$ and $\mathcal{E}$ in any way.

If we check the properties guaranteed by the fact that $\mathcal{U}$ is comparable to every $\mathcal{T}_i$, we may observe that $\mathcal{E}$ is an environment for $\mathcal{U}$ if $H_{\mathcal{U}} \cap A_{\mathcal{E}} = \emptyset$. Respecting this condition might require renaming all actions of $\mathcal{E}$. This is however not the case, as $H_{\mathcal{U}} \cap ((I_{\mathcal{E}} \cap O_{\mathcal{T}}) \cup (O_{\mathcal{E}} \cap I_{\mathcal{T}})) = \emptyset$ since $\mathcal{U}$ is comparable to every $\mathcal{T}_i$ and the internal actions of $\mathcal{U}$ are disjoint from its input and output actions. So, renaming all actions of $\mathcal{E}$ which are internal actions of $\mathcal{U}$ will never require to violate the restriction on f we stated in the last paragraph. $\Box$

3.3.7 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 3.48 *Two task-PIOAs $(\mathcal{P}_1, RA_1, RS_1)$ and $(\mathcal{P}_2, RA_2, RS_2)$ are* comparable *if:*

1. *$\mathcal{P}_1$ and $\mathcal{P}_2$ are comparable (have the same external signature) .*
2. *RA_1 and RA_2 contain exactly the same external tasks.*

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

Definition 3.49 *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 $\mathcal{T}_2$, $tdists(\mathcal{T}_1||\mathcal{E}) \subseteq tdists(\mathcal{T}_2||\mathcal{E})$.*

3.3.8 Simulation Relations

We now present our new simulation relation definition. Our definition differs from previous definitions for simulation relations for PIOAs, for example, those in [Segala95], in that it relates two *distributions on states*, rather than two states, or a state and a distribution. Also, our definition uses the task structure.

Like other definitions for simulations for PIOAs, our new definition includes a start condition and a step condition. However, our step condition is not the most obvious: Starting from two distributions ϵ_1 and ϵ_2, where $\epsilon_1 \; R \; \epsilon_2$, we end up with two distributions ϵ_1' and ϵ_2'. We do not require that $\epsilon_1' \; R \; \epsilon_2'$; instead, we require that ϵ_1' and ϵ_2' be decomposable into related distributions. To describe this decomposition, we use the expansion notion defined in Section 3.1.2.

Definition 3.50 *Let $\mathcal{T}_1 = (\mathcal{P}_1, RA_1, RS_1)$ and $\mathcal{P}_2 = (\mathcal{P}_2, RA_2, RS_2)$ be two comprable closed task-PIOAs. Let R be a relation on discrete distributions over finite execution fragments such that, if $\epsilon_1 \; R \; \epsilon_2$ then*

- *$tdist(\epsilon_1) = tdist(\epsilon_2)$.*
- *there exist equivalence classes $S_1 \in RS_1$ and $S_2 \in RS_2$ such that $supp(lstate(\epsilon_1)) \subseteq S_1$ and $supp(lstate(\epsilon_2)) \subseteq S_2$.*

Then we say that R is a simulation relation *from $\mathcal{T}_1$ to $\mathcal{T}_2$ if it satisfies the following properties:*

1. ***Start condition:*** *$\delta(\bar{q}_1) \; R \; \delta(\bar{q}_2)$.*
2. ***Step condition:*** *There exists a mapping $corrtasks : (RS_1 \times RA_1) \rightarrow {RA_2}^*$ such that the following holds: If $\epsilon_1 \; R \; \epsilon_2$ and T is a task of $\mathcal{T}_1$, then $\epsilon_1' \; \mathcal{E}(R) \; \epsilon_2'$, where $\epsilon_1' = apply(\epsilon_1, T)$ and $\epsilon_2' = apply(\epsilon_2, corrtasks([lstate(\epsilon_1)], T))$.*[1]

[1]That is, we apply the *corrtasks* function to the unique equivalence class of all the final states of the execution fragments in ϵ_1.

The following lemma gives a simple consequence of our simulation relation definition. The definition of a simulation relation says that any two R-related distributions on finite execution fragments must have the same trace distribution. This lemma extends this property by saying that any pair of distributions on execution fragments that are related by the expansion of the relation R, $\mathcal{E}(R)$, must also have the same trace distribution. (For the proof, the only property of simulation relations that we need is that related distributions on execution fragments have the same trace distribution.)

Lemma 3.51 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two closed task-PIOAs, R a simulation from $\mathcal{T}_1$ to $\mathcal{T}_2$. Let ϵ_1 and ϵ_2 be discrete distributions on finite execution fragments of $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively, such that $\epsilon_1\ \mathcal{E}(R)\ \epsilon_2$. Then $tdist(\epsilon_1) = tdist(\epsilon_2)$.*

Proof. Let η_1, η_2 and w be the measures and weighting functions used to prove that $\epsilon_1\ \mathcal{E}(R)\ \epsilon_2$. Since $\epsilon_1 = flatten(\eta_1)$, $tdist(\epsilon_1) = \sum_{\rho_1 \in supp(\eta_1)} \eta_1(\rho_1)\, tdist(\rho_1)$. Since w is a weighting function, we can rewrite the expression on the right as $\sum_{\rho_1 \in supp(\eta_1)} \sum_{\rho_2 \in supp(\eta_2)} w(\rho_1, \rho_2)\, tdist(\rho_1)$. Since $\rho_1\ R\ \rho_2$ whenever $w(\rho_1, \rho_2) > 0$, and since, by the definition of a simulation relation, $tdist(\rho_1) = tdist(\rho_2)$ whenever $\rho_1\ R\ \rho_2$, we can replace $tdist(\rho_1)$ by $tdist(\rho_2)$.
Thus, $tdist(\epsilon_1) = \sum_{\rho_1 \in supp(\eta_1)} \sum_{\rho_2 \in supp(\eta_2)} w(\rho_1, \rho_2)\, tdist(\rho_2)$. By exchanging sums, this last expression is equal to $\sum_{\rho_2 \in supp(\eta_2)} \sum_{\rho_1 \in supp(\eta_1)} w(\rho_1, \rho_2)\, tdist(\rho_2)$.

Now, since w is a weighting function, we can simplify the inner sum, thus getting $tdist(\epsilon_1) = \sum_{\rho_2 \in supp(\eta_2)} \eta_2(\rho_2)\, tdist(\rho_2)$. Since $\epsilon_2 = flatten(\eta_2)$, the right-hand side can be rewritten as $tdist(\epsilon_2)$. Thus, $tdist(\epsilon_1) = tdist(\epsilon_2)$, as needed. □

We now prove Theorem 3.53, which asserts that simulation relations are sound for showing inclusion of sets of trace distributions. We first give a lemma that provides the inductive step that we need for the proof of the main theorem.

Lemma 3.52 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two 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. 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 $\epsilon_2' = apply(\delta(\bar{q}_2), \rho_2\ corrtasks([lstate(\epsilon_1)], T))$*[2]
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 $\epsilon_2' = apply(\epsilon_2, corrtasks([lstate(\epsilon_1)], T))$. Recall that by Lemma 3.28, there is a single class S such that $supp(lstate(\epsilon_1)) \subseteq S$.

We apply Lemma 3.4: 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 $g(\epsilon) = apply(\epsilon, corrtasks([lstate(\epsilon_1)], T))$. We show that the hypothesis of Lemma 3.4 is satisfied, which implies that, by Lemma 3.4, $\epsilon_1'\ \mathcal{E}(R)\ \epsilon_2'$, as needed.

Distributivity of f and g follows directly by Proposition 3.32. 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 $\epsilon_1\ \mathcal{E}(R)\ \epsilon_2$, $\mu_1\ R\ \mu_2$. By the step condition for R, $apply(\mu_1, T)\ \mathcal{E}(R)\ apply(\mu_2, corrtasks([lstate(\mu_1)], T))$.

Observe that $apply(\mu_1, T) = f(\mu_1)$. We show that $apply(\mu_2, corrtasks([lstate(\mu_1)], T)) = g(\mu_2)$, which yields $f(\mu_1)\ \mathcal{E}(R)\ g(\mu_2)$. For this purpose, by definition of g, it suffices to show that $[lstate(\mu_1)] = [lstate(\epsilon_1)]$. Since $\epsilon_1 = flatten(\eta_1)$, and since, by the fact that $w(\mu_1, \mu_2) > 0$, $\eta_1(\mu_1) > 0$, $supp(\mu_1) \subseteq supp(\epsilon_1)$. Thus, $[lstate(\mu_1)] = [lstate(\epsilon_1)]$, as needed. □

The following theorem, Theorem 3.53, is the main soundness result. The proof simply puts the pieces together, using only Lemmas 3.42 (which says that the probabilistic execution generated by an infinite task scheduler can be seen as the limit of the probabilistic executions generated by some of the

[2]Lemma 3.28 implies that there is a single equivalence class S such that $supp(lstate(\epsilon_1)) \subseteq S$. Thus, ϵ_2' is well defined.

finite prefixes of the task scheduler), 3.52 (the step condition), 3.51 (related probabilistic executions have the same trace distribution), and 3.14 (limit commutes with *tdist*).

Theorem 3.53 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two comparable closed 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 scheduler, $T_1, T_2, \cdots$. For each $i > 0$, define ρ_i to be $corrtasks([lstate(apply(\bar{q}_1, T_1 \cdots T_{i-1}))], T_i)$. 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 3.42, 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 3.52 for the inductive step, for each $j \geq 0$, $\epsilon_{1,j}\ \mathcal{E}(R)\ \epsilon_{2,j}$. Then, by Lemma 3.51, for each $j \geq 0$, $tdist(\epsilon_{1,j}) = tdist(\epsilon_{2,j})$. By Lemma 3.14, $tdist(\epsilon_1) = \lim_{j\to\infty} tdist(\epsilon_{1,j})$, 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 3.53. 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 3.54 *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 $\mathcal{T}_2$, for which $|corrtasks(S,T)| \leq c$ for every S 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 3.53. □

The proofs presented in Sections 9-12 use a special case of the simulation relation definition, which we describe here.

Lemma 3.55 *Let $\mathcal{T}_1 = (\mathcal{P}_1, RA_1, RS_1)$ and $\mathcal{P}_2 = (\mathcal{P}_2, RA_2, RS_2)$ be two comparable closed task-PIOAs. Let R be a relation from discrete measures on finite execution fragments of $\mathcal{T}_1$ to discrete measures on finite execution fragments of $\mathcal{T}_2$ such that, if $\epsilon_1 R \epsilon_2$ then*

- *$tdist(\epsilon_1) = tdist(\epsilon_2)$.*
- *there exists equivalence classes $S_1 \in RS_1$ and $S_2 \in RS_2$ such that $supp(lstate(\epsilon_1)) \subseteq S_1$ and $supp(lstate(\epsilon_2)) \subseteq S_2$.*

Suppose further that the following conditions hold:

1. ***Start condition:*** *$\delta(\bar{q}_1)\ R\ \delta(\bar{q}_2)$.*
2. ***Step condition:*** *There exists a mapping $corrtasks : (RS_1 \times RA_1) \to {RA_2}^*$ such that, if $\epsilon_1 R \epsilon_2$ and T is a task of $\mathcal{T}_1$ that is enabled in $supp(lstate(\epsilon_1))$, then there exist*
    - *a probability measure p on a countable index set I,*
    - *probability measures ϵ'_{1j}, $j \in I$, on finite execution fragments of $\mathcal{P}_1$, and*
    - *probability measures ϵ'_{2j}, $j \in I$, on finite execution fragments 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*
- $\sum_{j\in I} p(j)(\epsilon'_{2j}) = apply(\epsilon_2, corrtasks([lstate(\epsilon_1)], T))$.

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

Proof. Straightforward, by Lemma 3.3. The additional enabling condition for T added here is not a serious restriction: for each non-enabled task T, we can make $corrtasks = \lambda$. □

3.4 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.

3.4.1 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 3.56 *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 state equivalence class S has a bit-string representation, which we denote by $\langle q \rangle$, $\langle tr \rangle$, and $\langle S \rangle$, respectively. The length of the bit-string representation of every action, state, transition, task, and state equivalence class 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, input task, output task, internal task, or state equivalence class. Also, there is a deterministic Turing machine that, given the representation of two candidate actions a_1 and a_2, decides whether $(a_1, a_2) \in RA$, and always runs in time at most b; similarly for the representation of two candidate states q_1 and q_2 and RS. 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.

3.4.2 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 3.57 *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. 3.56 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, and every state equivalence class of $\mathcal{T}_1 \| \mathcal{T}_2$ can be represented as a $3(b_1 + b_2)$-bit string, by combining the representations of state equivalence classes of both automata.

2. **Decoding:** It is possible to decide whether a candidate state $q = (q_1, q_2)$ is a state of $\mathcal{T}_1 \| \mathcal{T}_2$ 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 and for candidate state equivalence classes.

   It is possible to decide if a candidate input action is an input action of $\mathcal{T}_1 \| \mathcal{T}_2$ by checking if it is an input action of $\mathcal{T}_1$ 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 $\mathcal{T}_1 \| \mathcal{T}_2$ 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 input, 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 $(a_1, a_2) \in RA_{\mathcal{T}_1 \| \mathcal{T}_2}$ by checking if $(a_1, a_2) \in RA_{\mathcal{T}_1}$ or $(a_1, a_2) \in RA_{\mathcal{T}_2}$. Given two candidate states q and q' of $\mathcal{T}_1 \| \mathcal{T}_2$, it is possible to decide whether $(q, q') \in RS_{\mathcal{T}_1 \| \mathcal{T}_2}$ by checking if $(q \lceil \mathcal{T}_1, q' \lceil \mathcal{T}_1) \in RS_{\mathcal{T}_1}$ and $(q \lceil \mathcal{T}_2, q' \lceil \mathcal{T}_2) \in RS_{\mathcal{T}_2}$ (this restriction notation is defined after Definition 3.43). 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. 3.56 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. 3.56 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 3.57.

3.4.3 Hiding

Lemma 3.58 *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 3.58.

3.4.4 Time-Bounded Task Scheduler

Definition 3.59 *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.*

3.4.5 Implementation

In Section 3.3.7, 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 3.60 *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 3.61 *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 3.62 *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 $\mathcal{T}_2$. Then by Lemma 3.47, we obtain another environment $\mathcal{E}'$ for $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$, such that $\mathcal{E}\|\mathcal{T}_1$ is isomorphic to $\mathcal{E}'\|\mathcal{T}_1$ and $\mathcal{E}\|\mathcal{T}_3$ is isomorphic to $\mathcal{E}'\|\mathcal{T}_3$. We then apply case 1 to $\mathcal{E}'$, obtaining a b_3-time-bounded task scheduler ρ_3 for $\mathcal{T}_3$ such that

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

The isomorphism implies that

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

and

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

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 3.63 *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 3.57 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 3.64 *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 $\mathcal{T}_2$. □

3.4.6 Simulation Relations

The simulation relation we defined in Section 3.3.8 can be applied to time-bounded task-PIOAs. We obtain the following additional soundness theorem:

Theorem 3.65 *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 $\mathcal{T}_2 \| \mathcal{E}$, for which $|corrtasks(S, T)| \leq c$ for every S and T.*
Then $\mathcal{T}_1 \leq_{0,b,b_1,cb_1} \mathcal{T}_2$.

Proof. By Theorem 3.54 and the definition of our new implementation relationship. □

3.5 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.

3.5.1 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 3.66 *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 $\mathcal{T}_k$.

3.5.2 Time-Bounded Task-PIOA Families

Definition 3.67 *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 3.68 *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, input task, output task, internal task, or state equivalence class. Also, there is a deterministic Turing machine that, given k and two*

candidate actions a_1 and a_2, decides whether $(a_1, a_2) \in RA_{\mathcal{T}_k}$, and always runs in time at most $b(k)$; similarly for two candidate states q_1 and q_2, k and $RS_{\mathcal{T}_k}$. 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 3.69 *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 3.57 and the definition of a time-bounded task-PIOA family. □

Lemma 3.70 *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 3.58. □

Definition 3.71 *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 3.72 *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 3.73 *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 3.72 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 3.62 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 3.72 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 3.74 *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 3.72, 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 3.63 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 3.72 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 3.75 *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 3.64. □

Finally, we obtain a soundness result for simulation relations:

Theorem 3.76 *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 $(\mathcal{T}_2)_k \| \mathcal{E}$, for which $|corrtasks(S,T)| \leq c$ for every S and T.*
Then $\overline{\mathcal{T}}_1 \leq_{0,b,b_1,cb_1} \overline{\mathcal{T}}_2$.

Proof. By Theorem 3.65. □

3.5.3 Polynomial-Time Task-PIOA Families

Definition 3.77 *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 3.78 *Suppose $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ are two compatible polynomial time 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 3.57, 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 3.79 *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 3.70. □

Definition 3.80 *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 3.81 *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 3.82 *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 3.83 *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 3.73 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 3.84 *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 3.74 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 3.85 *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 3.75. □

And we have another soundness result for simulation relations:

Theorem 3.86 *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 $(\mathcal{T}_2)_k \| \mathcal{E}$, for which $|corrtasks(S,T)| \leq c$ for every S and T.
Then $\overline{\mathcal{T}}_1 \leq_{neg,pt} \overline{\mathcal{T}}_2$.

Proof. By Theorem 3.76. □

4 Ideal Systems for Oblivious Transfer

Having developed the basic machinery, we are ready to tackle our example. 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. The definitions are based on Canetti's definition of Oblivious Transfer in the Universal Composability framework [C01].

We parameterize our ideal systems by a set $C \subseteq \{Trans, Rec\}$, which indicates the corrupted endpoints. The system consists of two interacting task-PIOAs: the Functionality $Funct(C)$ and the Simulator $Sim(C)$, given in Figures 1 and 2.

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.

4.1 The Oblivious Transfer Functionality

$Funct(C)$ has two endpoints corresponding to $Trans$ and Rec. $Funct(C)$ receives in inputs at both endpoints. If $Rec \in C$, then $Funct(C)$ produces out' outputs at the Rec endpoint, which are inputs to $Sim(C)$, Otherwise, it produces out outputs, which are not inputs to $Sim(C)$. Task-PIOA $Funct$ is defined in Figure 1.

$Funct(C)$:

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

State relation: q_1 and q_2 are related iff:
$q_1.inval(Trans) = \perp$ iff $q_2.inval(Trans) = \perp$, and $q_1.inval(Rec) = \perp$ iff $q_2.inval(Rec) = \perp$.

Figure 1: The Functionality, $Funct(C)$

4.2 The Simulator

$Sim(C)$ is an arbitrary task-PIOA satisfying certain constraints. $Sim(C)$ 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 2.

Signature:

Input:
- if $Trans \in C$ then
  - $in(x)_{Trans}$, $x \in (\{0,1\} \rightarrow \{0,1\})$
- 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 $Trans \in C$ then $\{in(*)_{Trans}\}$.
If $Rec \in C$ then $\{in(*)_{Rec}\}$, $\{out'(*)_{Rec}\}$, $\{out(*)_{Rec}\}$.
Arbitrary tasks for other actions.

State relation: Arbitrary, subject to the consistency requirements given in Definition 3.25.

Figure 2: Constraints on $Sim(C)$

4.3 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 all out' actions.

5 Random Source Automata

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 3. Note that the equivalence classes obliterate distinctions based on the particular randomly chosen values.

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 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 $\{(1, f) : f \in Tdp\} \cup \{(2, z) : z \in (\{0,1\} \rightarrow D)\} \cup \{(3, b) : b \in (\{0,1\} \rightarrow \{0,1\})\}$.

$Src(D, \mu)$:

Signature:

Input:
  none

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

Internal:
  $choose - rand$

State:
  $chosenval \in D \cup \{\bot\}$, initially $\bot$

Transitions:

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

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

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

State relation: q_1 and q_2 are related iff:
$q_1.chosenval = \bot$ iff $q_2.chosenval = \bot$.

Figure 3: Code for $Src(D, \mu)$

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.

6.1 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 4.

Lemma 6.1 *In every reachable state of $Trans(D, Tdp)$: If $bval \neq \bot$ then $tdpp \neq \bot$, $zval \neq \bot$, $inval \neq \bot$, and $\forall i \in \{0,1\}, bval(i) = B(tdpp.inverse(zval(i))) \oplus inval(i)$.*

6.2 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 5.

Lemma 6.2 *In every reachable state of $Rec(D, Tdp, C)$:*

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

$Trans(D, Tdp)$:

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: $\{in(*)_{Trans}\}$, $\{rand(*)_{tdpp}\}$, $\{send(1,*)_{Trans}\}$, $\{receive(2,*)_{Trans}\}$, $\{send(3,*)_{Trans}\}$, $\{fix-bval_{Trans}\}$.

State relation: q_1 and q_2 are related iff:
$q_1.inval = \perp$ iff $q_2.inval = \perp$, $q_1.tdpp = \perp$ iff $q_2.tdpp = \perp$, $q_1.zval = \perp$ iff $q_2.zval = \perp$, and $q_1.bval = \perp$ iff $q_2.bval = \perp$.

Figure 4: Code for $Trans(D, Tdp)$

6.3 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 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

$Rec(D, Tdp, C)$:

Signature:

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

Output:
- $send(2,z)_{Rec}$, $z \in (\{0,1\} \rightarrow D)$
- 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 $\perp$
- $yval, zval \in (\{0,1\} \rightarrow D) \cup \{\perp\}$, 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:

$\{in(*)_{Rec}\}$, $\{rand(*)_{yval}\}$, $\{receive(1,*)_{Rec}\}$, $\{send(2,*)_{Rec}\}$, $\{receive(3,*)_{Rec}\}$, $\{fix - zval_{Rec}\}$.
If $Rec \in C$ then $\{out(*)_{Rec}\}$.
If $Rec \notin C$ then $\{out'(*)_{Rec}\}$.

State relation: q_1 and q_2 are related iff:
$q_1.inval = \perp$ iff $q_2.inval = \perp$, and similarly for tdp, $yval$, $zval$, and $outval$.

Figure 5: Code for $Rec(D, Tdp, C)$

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 6 and 7 .

6.4 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.

$Adv(D, Tdp, C)$:

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

Lemma 6.3 *In every reachable state of RS the following hold:*

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

Lemma 6.4 *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 $Trans(D, Tdp).bval = b$.*

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

State relation: Arbitrary RS, subject to consistency requirements, and such that there exists an equivalence relation RN on the valuations of the new variables where $(q_1, q_2) \in RS$ iff:

1. There is a bijection between $q_1.messages$ and $q_2.messages$, such that if $((i_1, m_1), p_1) \in q_1.messages$ corresponds to $((i_2, m_2), p_2) \in q_2.messages$ then $i_1 = i_2$ and $p_1 = p_2$.
2. If $Rec \in C$ then $q_1.outval(Rec) = \bot$ iff $q_2.outval(Rec) = \bot$.
3. The valuations on new variables in q_1 and q_2 are RN-related.

Figure 7: Code for $Adv(D, Tdp, C)$ (Part II)

5. *If $Rec.tdp = f \neq \bot$ then*
   (a) *$Adv.messages$ contains $(1, f)$.*
   (b) *$Trans.tdpp \neq \bot$ and $Trans.tdpp.funct = f$.*
6. *If $Rec.zval = z \neq \bot$ then $Rec.yval \neq \bot$, $Rec.inval \neq \bot$, $Rec.tdp \neq \bot$, $z(Rec.inval) = Rec.tdp(Rec.yval(Rec.inval))$, and $z(1 - Rec.inval) = Rec.yval(1 - Rec.inval)$.*
7. *If $Trans.zval = z \neq \bot$ then*
   (a) *$Adv.messages$ contains $(2, z)$.*
   (b) *$Rec.zval = z$.*
8. *If $Trans.bval = b \neq \bot$ then*
   (a) *$Trans.tdpp \neq \bot$, $Trans.zval \neq \bot$, $Trans.inval \neq \bot$, and $i \in \{0, 1\}$,*
   $b(i) = B(Trans.tdpp.inverse(Trans.zval(i))) \oplus Trans.inval(i)$.
   (b) *$Rec.inval \neq \bot$ and for $i = Rec.inval$, $b(i) = B(Rec.yval(i)) \oplus Trans.inval(i)$.*
9. *If $Rec.outval = x \neq \bot$ then*
   (a) *$x = Trans.bval(Rec.inval) \oplus B(Rec.yval(Rec.inval))$.*
   (b) *$x = Trans.inval(Rec.inval)$.*
10. *If $Trans.tdpp \neq \bot$ and $Trans.zval \neq \bot$, then $Rec.yval \neq \bot$, $Rec.inval \neq \bot$, and in addition $Trans.tdpp.inverse(Trans.zval(Rec.inval)) = Rec.yval(Rec.inval)$.*

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

Lemma 6.5 *If $C = \{Rec\}$ then, in every reachable state of $RS(D, Tdp, C)$, the following holds:*

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

7 The Main Theorems

In this section, we state the main theorem of this paper. It is really four theorems, for the four possible sets of corrupted parties.

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

7.1 Families of Sets

We assume two families of sets:

- $\overline{D} = \{D_k\}_{k \in \mathsf{N}}$, a family of finite domains. For example, D_k might be the set of length k bit strings.
- $\overline{Tdp} = \{Tdp_k\}_{k \in \mathsf{N}}$, a family of sets of trap-door permutations such that the domain of $f \in Tdp_k$ is D_k.

We also define the following derived families of sets:

- $\overline{Tdpp} = \{Tdpp_k\}_{k \in \mathsf{N}}$, 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.
- $\overline{M} = \{M_k\}_{k \in \mathsf{N}}$, a family of message alphabets, where $M_k = \{(1, f) : f \in Tdp_k\} \cup \{(2, z) : z \in (\{0,1\} \to D_k)\} \cup \{(3, b) : b \in (\{0,1\} \to \{0,1\})\}$.

7.2 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 $\{RS_k\}_{k \in \mathsf{N}}$, where, for each k, RS_k is a real system with parameters (D_k, Tdp_k, C). Thus, $RS_k = Trans(D_k, Tdp_k) \| Rec(D_k, Tdp_k, C) \| Src(Tdpp_k)_{tdpp} \| Src(\{0,1\} \to D_k)_{yval} \| Adv_k$, 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 $\{IS_k\}_{k \in \mathsf{N}}$, 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 $Sim(C)$.

7.3 Theorem Statements

In the following theorem, the four possible values of C yield four theorems, which we prove in Sections 9, 10, 11, and 12, respectively.

Theorem 7.1 *For every $C \subseteq \{Trans, Rec\}$ 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 $\overline{RS} \leq_{neg,pt} \overline{IS}$.

8 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.

8.1 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:

Definition 8.1 *A* hard-core predicate *for $\overline{D}$ and $\overline{Tdp}$ is a predicate $B : \bigcup_{k \in \mathsf{N}} D_k \to \{0,1\}$, such that*

1. *B is polynomial-time computable.*
2. *For every probabilistic polynomial-time non-uniform predicate $G = \{G_k\}_{k \in \mathsf{N}}$,[3] there is a negligible function ϵ such that, for all k,*

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

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 [Foundations of Cryptography, Volume I Basic Tools, by Oded Goldreich, Cambridge University Press, 2001, reprint of 2003, p. 64.] [GOLDREICH03].

8.2 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.

With these two PIOA families, Definition 8.1 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 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 8.2 *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 $Src(Tdp_k)$,*
- *$\overline{Src_{yval}} = \{(Src_{yval})_k\}_{k \in \mathsf{N}}$, 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. 8.*

Definition 8.3 *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 $Src(\{0,1\})$.*

Definition 8.4 *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 $\overline{SH} \leq_{neg,pt} \overline{SHR}$.*

[3] 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 $p(k)$.

$H(D, Tdp, B)$:

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

State relation: q_1 and q_2 are related iff:

$q_1.fval = \perp$ iff $q_2.fval = \perp$, and similarly for $yval$, $zval$, and $bval$.

Figure 8: Hard-core predicate automaton, $H(D, Tdp, B)$

We show that this definition is equivalent to Definition 8.1 by means of the following two theorems.

Theorem 8.5 *If B is a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ according to Definition 8.1, then B is also a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ according to Definition 8.4.*

Proof. Suppose that B is a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ according to Definition 8.1. Definition 8.1 implies that B is polynomial-time computable, which is required by Definition 8.4.

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 $\{choose-bval\}$ task of $(Src_{bval})_k$.
3. Replace each occurrence of the $\{fix-zval\}$ task of H_k with the $\{choose-zval\}$ 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 $SH_k \| \mathcal{E}$,

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

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 8.1. 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 a task of $(\mathcal{E}_{max})_k$ and goes on to the next task if it is not.
   - If T is an output or internal 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 an input task of $(\mathcal{E}_{max})_k$ of the form $\{rand(*)_{tdp}\}$ then G_k simulates the action $rand(f)_{tdp}$, where f is G_k's first input argument. Similarly, if T is of the form $\{rand(*)_{zval}\}$ 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 $\{rand(*)_{bval}\}$ 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 8.1 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 8.6 *If B is a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ according to Definition 8.4, then B is also a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ according to Definition 8.1.*

Proof. Suppose that B is a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ according to Definition 8.4. Definition 8.4 implies that B is polynomial-time computable, which is required by Definition 8.1.

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. 9.

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

Signature:

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

Output:
$accept$

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

Transitions:

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

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

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

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

Tasks: $\{rand(*)_{tdp}\}$, $\{rand(*)_{zval}\}$, $\{rand(*)_{bval}\}$, $\{accept\}$.
State relation: q_1 and q_2 are related iff:
$q_1.fval = \bot$ iff $q_2.fval = \bot$, and similarly for $zval$, and $bval$.

Figure 9: Environment evaluating the G predicate, $\mathcal{E}(G)(D, Tdp, B)$

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 8.4 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:

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

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. 8.4 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. □

8.3 Consequences of the New Definition

In this subsection, we formulate in our framework two important consequences that follow from our new definition of a hard-core predicate, and that are used in our analysis of the Oblivious Transfer algorithm. The first one says 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, 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.

The second consequence says that, if the results of applying a hard-core predicate are combined with inputs from the environment using $\oplus$, the final results still look random to the environment. This fact is needed because, in the protocol, the transmitter computes and sends $B(f^{-1}(zval(i))) \oplus inval(i)$, $i \in \{0, 1\}$, rather than just $B(f^{-1}(zval(i)))$.

8.3.1 Applying a Hard-Core Predicate Twice

Here, 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 8.4.

Definition 8.7 *The task-PIOA family $\overline{SH2}$ is defined as $hide_{\{rand(y)_{yval0}, rand(y)_{yval1}\}}(\overline{Src_{tdp}} \| \overline{Src_{yval0}} \| \overline{Src_{yval1}} \| \overline{H0} \| \overline{H1})$, 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 8.8 *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 8.9 *If B is a hard-core predicate, then $\overline{SH2} \leq_{neg,pt} \overline{SHR2}$.*

Proof. By Theorem 8.5, 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 $hide_{rand(y)_{yval0}}(\overline{Src_{tdp}} \| \overline{Src_{yval0}} \| \overline{H0} \| \overline{Src_{zval1}} \| \overline{Src_{bval1}})$, 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 8.1 implies that

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

This is because these two systems are simple renamings of the $\overline{SH}$ and $\overline{SHR}$ systems described in Section 8.2.

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 3.74 implies that

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

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 8.1 implies that

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

Now let $\overline{I}$ be the polynomial-time-bounded task-PIOA family $\overline{Src_{zval1}} \| \overline{Src_{bval1}}$. Then Lemma 3.74 implies that

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

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 3.83) implies that $\overline{SH2} \leq_{neg,pt} \overline{SHR2}$. □

8.3.2 Combining two hard-core bits with two input values

This material is for Case 1 of the proof.

In the Oblivious Transfer protocol, the transmitter does not send $B(f^{-1}(z_i))$ ($i \in \{0,1\}$), but rather, $B(f^{-1}(z_i)) \oplus x_i$, where x_i is an input bit provided by the environment. In this subsection, we prove that the resulting bits still look random to a polynomial-time-bounded environment. This result follows from the fact that B is a hard-core predicate, without requiring any new computational assumptions.

For this purpose, we add an interface that efficiently reduces any instance of the computational problem specific to our protocol into an instance of the computational problem we examined in the previous subsection. Specifically, we define a new polynomial time-bounded task-PIOA family $\overline{Ifc} = \{Ifc_k\}_{k \in \mathsf{N}}$. Task-PIOA Ifc_k receives, as inputs, a trapdoor permitation f, two bits b_0 and b_1, two elements, z_0 and z_1, of D_k, and a pair of bits (x_0, x_1), and outputs the same trapdoor permutation f, the two pairs (z_0, z_1), and $(b_0 \oplus x_0, b_1 \oplus x_1)$. Interface automaton Ifc_k is defined to be $Ifc(Tdp_k, D_k)$, where $Ifc(Tdp, D)$ is defined in Fig. 10.

Now we define $\overline{SHOT}$ and $\overline{SHROT}$, two task-PIOA families that we will need to compare in our proofs in Section 9.5.

Definition 8.10 *Consider the task-set family $\overline{U} = \{U_k\}_{k \in \mathsf{N}}$, where U_k is the set $\{\{rand(*)_{tdp}\}, \{rand(*)_{zval0}\}, \{rand(*)_{zval1}\}, \{rand(*)_{bval0}\}, \{rand(*)_{bval1}\}\}$ of tasks of $SH2_k$ and $SHR2_k$. The task-PIOA family $\overline{SHOT} = \{SHOT_k\}_{k \in \mathsf{N}}$ is defined as $hide_{\overline{U}}(\overline{SH2} \| \overline{Ifc})$. Also, the task-PIOA family $\overline{SHROT} = \{SHROT_k\}_{k \in \mathsf{N}}$ is defined as $hide_{\overline{U}}(\overline{SHR2} \| \overline{Ifc})$.*

Lemma 8.11 *$\overline{SHOT} \leq_{neg,pt} \overline{SHROT}$.*

Proof. By Lemma 8.9, $\overline{SH2} \leq_{neg,pt} \overline{SHR2}$. The task-PIOA family $\overline{Ifc}$ is 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 3.74),

$$\overline{SH2} \| \overline{Ifc} \leq_{neg,pt} \overline{SHR2} \| \overline{Ifc}.$$

Then, since hiding output tasks of polynomial-time-bounded task-PIOA families preserves the $\leq_{neg,pt}$ relation (Lemma 3.85), we have that

$$hide_{\overline{U}}(\overline{SH2} \| \overline{Ifc}) \leq_{neg,pt} hide_{\overline{U}}(\overline{SHR2} \| \overline{Ifc}),$$

which in turn implies that $\overline{SHOT} \leq_{neg,pt} \overline{SHROT}$. □

Some invariants will be helpful in the later proofs:

Lemma 8.12 *In all reachable states of $SHOT$:*

$Ifc(Tdp, D)$:

Signature:

Input:

$rand(f)_{tdp}$, $f \in Tdp$
$rand(z)_{zval0}$, $rand(z)_{zval1}$, $z \in D$
$rand(b)_{bval0}$, $rand(b)_{bval1}$, $b \in \{0,1\}$
$in(x)_{Trans}$, $x \in \{0,1\} \to \{0,1\}$

Output:

$send(1, f)_{Trans}$, $f \in Tdp$
$send(2, z)_{Rec}$, $z \in \{0,1\} \to D$
$send(3, b)_{Trans}$, $b \in \{0,1\} \to \{0,1\}$

Internal:

$fix - bxorx$

State:

$fval \in Tdp \cup \bot$, initially $\bot$,
$zval \in \{0,1\} \to (D \cup \bot)$, initially identically $\bot$
$bval \in \{0,1\} \to \{0,1,\bot\}$, initially identically $\bot$
$xval, bxorx \in (\{0,1\} \to \{0,1\}) \cup \{\bot\}$, initially $\bot$

Transitions:

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

$rand(z)_{zvali}$, $i \in \{0,1\}$
Effect:
if $zval(i) = \bot$ then $zval(i) := z$

$rand(b)_{bvali}$, $i \in \{0,1\}$
Effect:
if $bval(i) = \bot$ then $bval(i) := b$

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

$fix - bxorx$
Precondition:
$\forall i \in \{0,1\}$, $bval(i) \neq \bot$
$xval, fval, zval \neq \bot$
$bxorx = \bot$
Effect:
for $i \in \{0,1\}$ do
$bxorx(i) := bval(i) \oplus xval(i)$

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

$send(2, z)_{Rec}$
Precondition:
$zval(i) \neq \bot$ $(i \in \{0,1\})$
$z = zval$
Effect:
none

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

Tasks: $\{rand(*)_{tdp}\}$, $\{rand(*)_{zval0}\}$, $\{rand(*)_{zval1}\}$, $\{rand(*)_{bval0}\}$, $\{rand(*)_{bval1}\}$, $\{in(*)_{Trans}\}$, $\{fix - bxorx\}$, $\{send(1, f)_{Trans}\}$, $\{send(2, z)_{Rec}\}$, $\{send(3, b)_{Trans}\}$.

State relation: q_1 and q_2 are related iff:
$q_1.fval = \bot$ iff $q_2.fval = \bot$, and similarly for $xval$ and $bxorx$;
$\forall i \in \{0,1\}$, $q_1.bval(i) = \bot$ iff $q_2.bval(i) = \bot$, and similarly for $zval(i)$.

Figure 10: Interface, $Ifc(Tdp, D)$

1. *$Ifc.fval = H0.fval = H1.fval$.*
2. *If $Ifc.fval \neq \bot$ then $Ifc.fval = Src_{tdp}.chosenval$.*
3. *If $Hi.yval \neq \bot$ then $Hi.yval = Src_{yvali}.chosenval$.*
4. *If $Hi.zval \neq \bot$ then $Hi.yval \neq \bot$, $Hi.fval \neq \bot$, and $Hi.xval = Hi.fval(Hi.yval)$.*
5. *If $Hi.bval \neq \bot$ then $Hi.yval \neq \bot$ and $Hi.bval = B(Hi.yval)$.*
6. *If $Ifc.bval(i) \neq \bot$ then $Ifc.bval(i) = Hi.bval$.*

7. *If* $Ifc.bxorx \neq \perp$ *then* $Ifc.xval \neq \perp$, *for* $i \in \{0,1\}$, $Ifc.bval(i) \neq \perp$, *and for* $i \in \{0,1\}$, $Ifc.bxorx(i) = Ifc.bval(i) \oplus Ifc.xval(i)$.

Lemma 8.13 *In all reachable states of* $SHROT$*:*

1. *If* $Ifc.fval \neq \perp$ *then* $Ifc.fval = Src_{tdp}.chosenval$.
2. *If* $Ifc.zval(0) \neq \perp$ *then* $Ifc.zval(0) = Src_{zval0}.chosenval$.
3. *If* $Ifc.zval(1) \neq \perp$ *then* $Ifc.zval(1) = Src_{zval1}.chosenval$.
4. *If* $Ifc.bval(0) \neq \perp$ *then* $Ifc.bval(0) = Src_{bval0}.chosenval$.
5. *If* $Ifc.bval(1) \neq \perp$ *then* $Ifc.bval(1) = Src_{bval1}.chosenval$.
6. *If* $Ifc.bxorx \neq \perp$ *then* $Ifc.xval \neq \perp$, *for* $i \in \{0,1\}$, $Ifc.bval(i) \neq \perp$, *and for* $i \in \{0,1\}$, $Ifc.bxorx(i) = Ifc.bval(i) \oplus Ifc.xval(i)$.

8.3.3 Combining a single hard-core bit with an input value

For Case 2 of the proof, we define new $\overline{SHOT'}$ and $\overline{SHROT'}$ families.

Definition 8.14 *The task-PIOA family* $\overline{SHOT'}$ *is defined as*

$$hide_{\{rand(*)_{tdp},rand(*)_{zval},rand(*)_{bval},rand(*)_{yval'}\}}(\overline{SH}\|\overline{Src_{yval'}}\|\overline{Ifc'}),$$

where

- $\overline{SH}$ *is given in Def. 8.2,*
- $\overline{Src_{yval'}} = \{(Src_{yval'})_k\}_{k\in\mathsf{N}}$, *where each* $(Src_{yval'})_k$ *is isomorphic to* $Src(D_k)$,
- $\overline{Ifc'}$ *is defined in Fig. 11 and 12.*

Definition 8.15 *The task-PIOA family* $\overline{SHROT'}$ *is defined as*

$$hide_{\{rand(*)_{tdp},rand(*)_{zval},rand(*)_{bval},rand(*)_{yval'}\}}(\overline{SHR}\|\overline{Src_{yval'}}\|\overline{Ifc'}),$$

where $\overline{SHR}$ *is given in Def. 8.3 while* $\overline{Src_{yval'}}$ *and* $\overline{Ifc'}$ *are as in Def. 8.14.*

Again, we have:

Lemma 8.16 $\overline{SHOT'} \leq_{neg,pt} \overline{SHROT'}$.

Proof. By Definition 8.4, $\overline{SH} \leq_{neg,pt} \overline{SHR}$. 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 3.84),

$$\overline{SH}\|\overline{Ifc'}\|\overline{Src_{yval'}} \leq_{neg,pt} \overline{SHR}\|\overline{Ifc'}\|\overline{Src_{yval'}}.$$

Now, if we define $\overline{U} = \{rand(*)_{tdp}, rand(*)_{zval}, rand(*)_{bval}, rand(*)_{yval'}\}$, we have that

$$hide_{\overline{U}}(\overline{SH}\|\overline{Ifc'}\|\overline{Src_{yval'}}) \leq_{neg,pt} hide_{\overline{U}}(\overline{SHR}\|\overline{Ifc'}\|\overline{Src_{yval'}}),$$

since hiding output tasks of polynomial-time-bounded task-PIOA families preserves the $\leq_{neg,pt}$ relation (Lemma 3.85).

This is equivalent to say that $\overline{SHOT'} \leq_{neg,pt} \overline{SHROT'}$. □

Some invariants will be helpful in the later proofs:

$Ifc'(Tdp, D)$:

Signature:

Input:

$rand(f)_{tdp}$, $f \in Tdp$
$rand(z)_{zval}$, $z \in D$
$rand(y)_{yval'}$, $y \in D$
$rand(b)_{bval}$, $b \in \{0,1\}$
$in(x)_{Trans}$, $x \in \{0,1\} \rightarrow \{0,1\}$
$in(i)_{Rec}$, $i \in \{0,1\}$
$out'(x)_{Rec}$, $x \in \{0,1\}$

Output:

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

Internal:

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

State:

$fval \in (Tdp \cup \perp)$, initially $\perp$,
$zval' \in (D \cup \perp)$, initially $\perp$
$yval' \in (D \cup \perp)$, initially $\perp$
$zval \in (\{0,1\} \rightarrow D) \cup \{\perp\}$, initially $\perp$
$bval' \in \{0,1,\perp\}$, initially $\perp$
$bval \in (\{0,1\} \rightarrow \{0,1\}) \cup \{\perp\}$, initially $\perp$
$inval(Trans), inval(Rec) \in \{0,1,\perp\}$, initially $\perp$
$inval2(Trans) \in (\{0,1\} \rightarrow \{0,1\}) \cup \{\perp\}$, initially $\perp$

Figure 11: Interface, $Ifc'(Tdp, D)$ (Part I)

Lemma 8.17 *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* $H.zval \neq \perp$ *then* $H.fval \neq \perp$, $H.yval \neq \perp$ *and* $H.zval = H.fval(H.yval)$.
6. *If* $Ifc'.zval' \neq \perp$ *then* $Ifc'.zval' = H.zval$.
7. *If* $H.bval \neq \perp$ *then* $H.yval \neq \perp$ *and* $H.bval = B(H.yval)$.
8. *If* $Ifc'.bval' \neq \perp$ *then* $Ifc'.bval' = H.bval$.
9. *If* $Ifc'.zval \neq \perp$ *then* $Ifc'.yval' \neq \perp$ *and* $Ifc'.bval' \neq \perp$.

Lemma 8.18 *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* $Ifc'.bval' = Src_{bval}.chosenval$.

9 Correctness Proof, Case 1: Neither Party Corrupted

To show correctness, we consider four cases, based on which parties are corrupted. This section is devoted to the case where neither party is corrupted, that is, where $C = \emptyset$, and Sections 10-12 deal with the other three cases.

$Ifc'(Tdp, D)$:

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 $bval' := b$

$fix - zval_{Rec}$
Precondition:
$yval', zval', bval', inval(Rec), fval \neq \perp$
$zval = \perp$
Effect:
$zval(inval(Rec)) := fval(yval')$
$zval(1 - inval(Rec)) := zval'$

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

$out''(x)_{Rec}$
Precondition:
$x = inval(Trans) \neq \perp$
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: $\{rand(*)_{tdp}\}$, $\{rand(*)_{yval'}\}$, $\{rand(*)_{zval}\}$, $\{rand(*)_{bval}\}$, $\{in(*)_{Trans}\}$, $\{in(*)_{Rec}\}$, $\{out'(*)_{Rec}\}$, $\{send(1,*)_{Trans}\}$, $\{send(2,*)_{Rec}\}$, $\{send(3,*)_{Trans}\}$, $\{out''(*)_{Rec}\}$, $\{fix - zval_{Rec}\}$, $\{fix - bval_{Trans}\}$.

State relation: q_1 and q_2 are related iff:
$q_1.fval = \perp$ iff $q_2.fval = \perp$, and similarly for $zval'$, $yval'$, $zval$, $bval$, $bval'$, $inval(Trans)$, $inval(Rec)$ and $inval2(Trans)$.

Figure 12: Interface, $Ifc'(Tdp, D)$ (Part II)

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

Since $C = \emptyset$ everywhere in this section, we drop explicit mention of C from now on in the section.

We begin by expressing 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 this structed simulator 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 8. The style of the proof that $\overline{Int1} \leq_{neg,pt} \overline{Int2}$ is an alternative to the usual "Distinguisher" arguments from the traditional cryptographic protocol literature. Our proof does not contain any arguments "by contradiction"; instead, it relies on positive results about implementation, composition, and hiding. The essential technical ideas that appear in the usual Distinguisher argument still appear in our argument, but in a direct (and systematic) way.

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 3.3.8.

The multi-level structure, with intermediate levels $\overline{Int1}$ and $\overline{Int2}$, is also used for the case where $C = \{R\}$, that is, where only the Receiver is corrupted. However, it is not needed for the other two cases, where $C = \{T\}$ and where $C = \{T, R\}$.

In the rest of this section, we fix a particular polynomial-time-bounded adversary family $\overline{Adv}$.

9.1 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*, and *rand* actions hidden.

- $TR(D_k, Tdp_k)$, an abstract combination of $Trans(D_k, Tdp_k)$ and $Rec(D_k, Tdp_k, \emptyset)$.
- $(Src(Tdpp_k)_{tdpp})_k$, isomorphic to $Src(Tdpp_k)$.
- $(Src(\{0,1\} \to D_k)_{zval})_k$, isomorphic to $Src(\{0,1\} \to D_k)$.
- $(Src(\{0,1\} \to \{0,1\})_{bval})_k$, isomorphic to $Src(\{0,1\} \to \{0,1\})$.
- Adv_k, the same adversary as in RS_k.

TR has *send* outputs that are inputs to *Adv*. *Adv*'s *receive* outputs are not connected to anything. *Adv* may also interact with the environment, using other inputs and outputs.

$TR(D, Tdp)$ is defined (for arbitrary parameters D and Tdp) in Figure 13. *TR* simply acquires, as input, a trap-door permutation pair, a pair of D values, and a pair of bits, and sends these in round 1, round 2, and round 3 messages respectively.

We define SIS_k, the structured ideal system, to be the composition $Funct_k \| SSim_k$.

Lemma 9.2 *In every reachable state of SIS_k:*

1. *$Adv_k.messages$ contains at most one round 1 message, at most one round 2 message, and at most one round 3 message.*

$TR(D, Tdp)$:

Signature:

Input:

$rand(p)_{tdpp}, p \in Tdpp$
$rand(z)_{zval}, z \in (\{0,1\} \to D)$
$rand(b)_{bval}, b \in (\{0,1\} \to \{0,1\})$

Output:

$send(1, f)_{Trans},\ f \in Tdp$
$send(2, z)_{Rec},\ z \in (\{0,1\} \to D)$
$send(3, b)_{Trans},\ b \in (\{0,1\} \to \{0,1\})$

State:

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

Transitions:

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

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

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

$send(1, f)_{Trans}$
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: $\{rand(*)_{tdpp}\}$, $\{rand(*)_{zval}\}$, $\{rand(*)_{bval}\}$, $\{send(1,*)_{Trans}\}$, $\{send(2,*)_{Rec}\}$, $\{send(3,*)_{Trans}\}$.

State relation: q_1 and q_2 are related iff:
$q_1.tdpp = \perp$ iff $q_2.tdpp = \perp$, and similarly for $zval$ and $bval$.

Figure 13: $TR(D, Tdp)$, for the case where $C = \emptyset$.

2. *If $Adv_k.messages$ contains $(1, f)$ then $TR_k.tdp = f$.*

3. *If $Adv_k.messages$ contains $(2, z)$ then $TR_k.zval = z$.*

4. *If $Adv_k.messages$ contains $(3, b)$ then $TR_k.bval = b$.*

5. *If $TR.bval \neq \perp$ then $TR.bval = Src_{bval}.chosenval$.*

Note that an ideal system IS_k consists of $Funct_k$ and some Sim_k satisfying the constraints defined in Figure 2. By definition, SIS_k is a specific system consisting of $Funct_k$ and a particular simulator, $SSim_k$, that satisfies those contraints. Therefore, to prove Theorem 9.1, it suffices to prove that $\overline{RS} \leq_{neg,pt} \overline{SIS}$.

9.2 *Int1*

We define system $Int1_k$ to be the same as SIS_k except that $TR(D_k, Tdp_k)$ is replaced by $TR1(D_k, Tdp_k)$. Code for $TR1(D, Tdp)$ appears in Figure 14. $TR1$ 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 $rand_{bval}$; rather, $TR1$ calculates $bval$ values using the hard-core predicate B and the inverse of the trap-door permutation applied to the $zval$ values, combined with the transmitter input values.

TR1(D, Tdp):

Signature:

Input:
$in(x)_{Trans}$, $x \in (\{0,1\} \rightarrow \{0,1\})$
$rand(p)_{tdpp}, p \in Tdpp$
$rand(z)_{zval}, z \in (\{0,1\} \rightarrow D)$

Output:
$send(1, f)_{Trans}$, $f \in Tdp$
$send(2, z)_{Rec}$, $z \in (\{0,1\} \rightarrow D)$
$send(3, b)_{Trans}$, $b \in (\{0,1\} \rightarrow \{0,1\})$

Internal:
$fix - bval_{Trans}$

State:
$inval(Trans) \in (\{0,1\} \rightarrow \{0,1\}) \cup \{\bot\}$, initially $\bot$
$tdpp \in Tdpp \cup \{\bot\}$, initially $\bot$
$zval \in (\{0,1\} \rightarrow D) \cup \{\bot\}$, initially $\bot$
$bval \in (\{0,1\} \rightarrow \{0,1\}) \cup \{\bot\}$, initially $\bot$

Transitions:

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

$rand(p)_{tdpp}$ or $rand(z)_{zval}$
Effect:
As for $TR(D, Tdp)$.

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

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

Tasks: $\{in(*)_{Trans}\}$, $\{rand(*)_{tdpp}\}$, $\{rand(*)_{zval}\}$, $\{send(1,*)_{Trans}\}$, $\{send(2,*)_{Rec}\}$, $\{send(3,*)_{Trans}\}$, $\{fix - bval_{Trans}\}$.

State relation: q_1 and q_2 are related iff:
$q_1.inval(Trans) = \bot$ iff $q_2.inval(Trans) = \bot$, and similarly for *tdpp*, *zval*, and *bval*.

Figure 14: *TR1*(D, Tdp), for the case where $C = \emptyset$.

Lemma 9.3 *In every reachable state of* $Int1_k$:

1. *If* $TR1.zval \neq \bot$ *then* $Src_{zval}.chosenval = TR1.zval$.

9.3 *Int2*

We define $Int2_k$ to be the same as SIS_k except that:

1. It includes a new random source $(Src(\{0,1\} \rightarrow \{0,1\})_{cval})_k$, which is isomorphic to $Src(\{0,1\} \rightarrow \{0,1\})$.
2. $TR(D_k, Tdp_k)$ is replaced by $TR2(D_k, Tdp_k)$, where $TR2(D, TDp)$ is identical to $TR1(D, Tdp)$ except that:
   (a) *TR2* includes an extra state variable $cval \in (\{0,1\} \rightarrow \{0,1\})$.
   (b) *TR2* has input action $rand(c)_{cval}$, which sets $cval := c$.
   (c) The line in $fix - bval_{Trans}$ in which the *bval* values are chosen is replaced by the line: for $i \in \{0,1\}$ do $bval(i) := cval(i) \oplus inval(i)$. That is, instead of calculating the *bval* values

using the hard-core predicate, *TR2* obtains them by applying $\oplus$ to two bits chosen randomly and the actual x inputs.

The code for $TR2(D, Tdp)$ appears in Figure 15.

$TR2(D, Tdp)$:

Signature:

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

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

Internal:
- $fix - bval_{Trans}$

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

Transitions:

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

$rand(p)_{tdpp}$ or $rand(z)_{zval}$
Effect:
  As for $TR(D, Tdp)$.

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

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

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

Tasks: $\{in(*)_{Trans}\}$, $\{rand(*)_{tdpp}\}$, $\{rand(*)_{zval}\}$, $\{rand(*)_{cval}\}$, $\{send(1,*)_{Trans}\}$, $\{send(2,*)_{Trans}\}$, $\{send(3,*)_{Trans}\}$, $\{fix - bval_{Trans}\}$.

State relation: q_1 and q_2 are related iff:
$q_1.inval(Trans) = \perp$ iff $q_2.inval(Trans) = \perp$, and similarly for *tdpp*, *zval*, and *cval*.

Figure 15: $TR2(D, Tdp)$, for the case where $C = \emptyset$.

Lemma 9.4 *In every reachable state of $Int2_k$:*

1. *If $TR2.cval \neq \perp$ then $TR2.cval = Src_{cval}.chosenval$.*

9.4 $\overline{RS}$ implements $\overline{Int1}$

We show:

Lemma 9.5 *For every k, $RS_k \leq_0 Int1_k$.*

We prove Lemma 9.5 by choosing an arbitrary environment *Env* for RS_k and $Int1_k$, and establishing a simulation relation from $RS_k \| Env$ to $Int1_k \| Env$. (See Section 3.3.6 for the definition of an environment.) Then we appeal to Theorem 3.55, the soundness result for simulation relations.

An interesting issue in proving Lemma 9.5 is in reconciling the different ways in which *zval* gets defined in *RS* and *Int1*. In *RS*, the choice is made in two steps, first choosing the *yval* values randomly and then calculating the *zval* values from the *yval* values, whereas in *Int1*, the *zval* values are chosen randomly, in one step.

We also show the following lemma, which is what we need to put the pieces of the proof together:

Lemma 9.6 $\overline{RS} \leq_{neg,pt} \overline{Int1}$.

Lemma 9.6 does not quite follow from Lemma 9.5. The reason is that the statement of Lemma 9.5 does not provide us with the needed bound on the growth of the length of the schedules. However, the simulation relation used to prove Lemma 9.5 does indeed guarantee such a bound; in fact, for each step of RS_k, the step correspondence yields at most two steps of $Int1_k$.

In the rest of this subsection, we fix *Env*, an environment for RS_k and $Int1_k$. We also suppress mention of k everywhere.

9.4.1 State correspondence

Here we define the correspondence R from states of $RS \| Env$ to states of $Int1 \| Env$, which we will show to be a simulation relation in Section 9.4.2.

In this mapping, most of the correspondences between variables are simple and direct. The one exception is the correspondences involving the randomly-chosen *zval* and *yval* values. Namely, in the *Int1* system, *zval* is chosen in one step, whereas in the *RS* system, *zval* is determined in three steps: first, *yval* is chosen randomly, then communicated to *Rec*, and then used to compute *zval*. We choose to allow the steps where *zval*'s value is determined to correspond at the two levels. Therefore, the states before *zval* is determined in the *Int1* system correspond to several kinds of states in the *RS* system, representing the different stages before *zval* is determined. In particular, before *zval* is determined in the *RS* system, a distribution on choices of *yval* in the *RS* system corresponds to "no choice" in the *Int1* system.

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

1. **Trace distribution equivalence:** $tdist(\epsilon_1) = tdist(\epsilon_2)$.
2. **State equivalence:** There exist state equivalence classes $S_1 \in RS_{RS \| Env}$ and $S_2 \in RS_{Int1 \| Env}$ such that $supp(lstate(\epsilon_1)) \subseteq S_1$ and $supp(lstate(\epsilon_2)) \subseteq S_2$.

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) $u.TR1.inval(Trans) = s.Trans.inval$.
   (d) $u.TR1.tdpp = s.Trans.tdpp$.
   (e) $u.TR1.zval = s.Rec.zval$.
   (f) $u.TR1.bval = s.Trans.bval$.
   (g) $u.Src_{tdpp} = s.Src_{tdpp}$.
   (h) $u.Src_{zval}.chosenval = s.Rec.zval$.
   (i) $u.Adv = s.Adv$.
   That is, the entire state is the same.
   (j) $u.Env = s.Env$.

2. For every $u \in supp(lstate(\epsilon_2))$:
   If $u.TR1.zval = \perp$ then one of the following holds:

   (a) For every $s \in supp(lstate(\epsilon_1))$, $s.Src_{yval}.chosenval = \perp$.
   That is, in all the states in the support of $lstate(\epsilon_1)$, $yval$ has not yet been chosen.

   (b) For every $s \in supp(lstate(\epsilon_1))$, $s.Rec.yval = \perp$, and $lstate(\epsilon_1).Src_{yval}.chosenval$ is the uniform distribution on $\{0,1\} \to D$.
   That is, in all the states in the support of $lstate(\epsilon_1)$, $yval$ has already been chosen by the Src_{yval}, but has not yet been output to Rec. Moreover, the values chosen by the Src form a uniform distribution.

   (c) $lstate(\epsilon_1).Rec.yval$ is the uniform distribution on $\{0,1\} \to D$.

9.4.2 The mapping proof

Lemma 9.7 *The relation R defined in Section 9.4.1 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, for every S, T, $|corrtasks(S,T)| \leq 2$.*

Proof. We prove that R satisfies the two conditions in Lemma 3.55.
Start condition: It is obvious that the Dirac measures on execution fragments consisting of the the unique start states s and u of $RS\|Env$ and $Int1\|Env$, respectively, are R-related. Property 1 of R holds because the state components of s and u on which R depends are all $\perp$. Property 2 of R holds because $s.Src_{yval}.chosenval = \perp$.
Step condition: We define $corrtasks(RS_{RS\|Env} \times RA_{RS\|Env}) \to RA^*_{Int1\|Env}$ as follows:
For any $(S,T) \in (RS_{RS\|Env} \times RA_{RS\|Env})$:

- If $T \in \{\{in(x)_{Trans}\}, \{in(i)_{Rec}\}, \{choose - rand_{tdpp}\}, \{rand_{tdpp}\}, \{fix - bval_{Trans}\}, \{send(1,f)_{Trans}\}, \{receive(1,f)_{Rec}\}, \{send(2,z)_{Rec}\}, \{receive(2,z)_{Trans}\}, \{send(3,b)_{Trans}\}, \{receive(3,b)_{Rec}\}$, or $\{out(x)_{Rec}\}\}$, then $corrtasks(S,T) = T$.
- If T is an output or internal task of Env or Adv that is not one of the tasks listed above, then $corrtasks(S,T) = T$.
- If $T \in \{\{choose - rand_{yval}\}, \{rand_{yval}\}\}$ then $corrtasks(S,T) = \lambda$ (the empty sequence).
- If $T = \{fix - zval_{Rec}\}$ then $corrtasks(S,T) = \{choose - rand_{zval}\}\ \{rand_{zval}\}$.

Suppose $(\epsilon_1, \epsilon_2) \in R$ and T is a task of $RS\|Env$ that is enabled in $supp(lstate(\epsilon_1))$. Let $\epsilon_1' = apply(\epsilon_1, T)$ and $\epsilon_2' = apply(\epsilon_2, corrtasks([lstate(\epsilon_1)], T))$.

The state equivalence property for ϵ_1 and ϵ_2 and Lemma 3.28 imply the state equivalence property for ϵ_1' and ϵ_2'; that is, there exist state equivalence classes $S_1 \in RS_{RS\|Env}$ and $S_2 \in RS_{Int1\|Env}$ such that $supp(lstate(\epsilon_1')) \subseteq S_1$ and $supp(lstate(\epsilon_2')) \subseteq S_2$.

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 1(j) of R.

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 1(i) of R.

Claim 2:

1. If T (defined above) is an output or internal task of *Env*, then

   (a) T is enabled in every state in $supp(lstate(\epsilon_2))$.
   To see this, fix any state $u \in supp(lstate(\epsilon_2))$; we show that T is enabled in u. Choose any $s \in supp(lstate(\epsilon_1))$. Since T is enabled in s and T is an output or internal task of *Env*, T is enabled in $s.Env$. Since, by Claim 1, $u.Env = s.Env$, T is enabled in $u.Env$, and hence in u, as needed.

   (b) There is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   By the next-action-determinism property for *Env*, we know that there is a unique action $a \in T$ that is enabled in q_{Env}. Since T is an output or internal task of *Env*, a is also the unique action in T that is enabled in each state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.

   (c) There is a unique transition of *Env* from q_{Env} with action a; let $tr_{Env} = (q_{Env}, a, \mu_{Env})$ be this transition.
   This follows from next-transition determinism for *Env*.

2. If T is an output or internal task of *Adv*, then

   (a) T is enabled in every state in $supp(lstate(\epsilon_2))$.
   By an argument analogous to the one for *Env*.

   (b) There is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.

   (c) 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 first show that the sequence of tasks $corrtasks([lstate(\epsilon_1)]], T)$ is enabled in every state in $supp(lstate(\epsilon_2))$. Then 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 execution fragments 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 fragment $\alpha \in supp(\epsilon'_1)$ is in $supp(\epsilon'_{1j})$ for a unique j; for every $j' \neq j$, $\epsilon'_{1j'}(\alpha) = 0$. (b) For each execution fragment $\alpha \in supp(\epsilon'_1)$, $\epsilon'_1(\alpha) = p(j)\epsilon'_{1j}(\alpha)$ 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.

To show that $(\epsilon'_{1j}, \epsilon'_{2j}) \in R$, we must establish Properties 1 and 2 of the definition of R for ϵ'_{1j} and ϵ'_{2j}. We must also show the trace distribution equivalence and state equivalence properties for ϵ'_{1j} and ϵ'_{2j}.

The state equivalence property follows for a generic reason: As noted above, there exist state equivalence classes $S_1 \in RS_{RS\|Env}$ and $S_2 \in RS_{Int1\|Env}$ such that $supp(lstate(\epsilon'_1)) \subseteq S_1$ and $supp(lstate(\epsilon'_2)) \subseteq S_2$. Since $supp(\epsilon'_{1j} \subseteq supp(\epsilon'_1)$ and $supp(\epsilon'_{2j} \subseteq supp(\epsilon'_2)$, it follows that $supp(lstate(\epsilon'_{1j})) \subseteq S_1$ and $supp(lstate(\epsilon'_{2j})) \subseteq S_2$. This implies state equivalence for ϵ'_{1j} and ϵ'_{2j}. Thus, we will not mention the state equivalence property within the individual proof cases.

We now proceed to consider the proof cases.

1. $T = \{in(x)_{Trans}\}$.

   Since T is an output task of Env, 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 $tr_{Env} = (q_{Env}, a, \mu_{Env})$ of Env from q_{Env} with action a. Here, $a = in(x)_{Trans}$ for a particular value of x.

   Next, 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 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 execution fragments $\alpha \in supp(\epsilon'_1)$ 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 Properties 1 and 2 of the definition 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 1(a) and 1(c), $u.Funct.inval(Trans) \neq \perp$ and $u.TR1.inval(Trans) \neq \perp$. 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 1(a) and 1(c), $u.Funct.inval(Trans) = u.TR1.inval(Trans) = \perp$. 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 $TR1.inval(Trans)$ in the $Int1$ system. It also updates the state of Env in both systems.

   We know by Property 1(a) that $u.Funct.inval(Trans) = s.Trans.inval$, by Property 1(c) that $u.TR1.inval(Trans) = s.Trans.inval$, and by Property 1(j) that $u.Env = s.Env$. By the effects of T in the definitions of $Trans$, $Funct$, and $TR1$, we know that $u'.Funct.inval(Trans) = s'.Trans.inval$, and $u'.TR1.inval(Trans) = s'.Trans.inval$; hence, Properties 1(a) and 1(c) hold for s' and u'. We also know that Property 1(j) holds for s' and u' by definition of ϵ'_{1j} and ϵ'_{2j}: in both s' and u', the state of Env is q_j. Since no state component other than $Trans.inval$ and Env in the RS system, and $Funct.inval(Trans)$, $TR1.inval(Trans)$, 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'_{1j} and ϵ'_{2j}.

   To establish Property 2, consider any state $u' \in supp(lstate(\epsilon'_{2j}))$ such that $u'.TR1.zval = \perp$. We need to show that one of the following holds:

   (a) For every $s' \in supp(lstate(\epsilon'_{1j}))$, $s'.Src_{yval}.chosenval = \perp$.
   (b) For every $s' \in supp(lstate(\epsilon'_{1j}))$, $s'.Rec.yval = \perp$, and $lstate(\epsilon'_{1j}).Src_{yval}.chosenval$ is the uniform distribution on $\{0,1\} \rightarrow D$.
   (c) $lstate(\epsilon'_{1j}).Rec.yval$ is the uniform distribution on $\{0,1\} \rightarrow D$.

   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 the effects of T, we know that $u.TR1.zval = u'.TR1.zval = \perp$. Then, by Property 2 for u, one of the following holds:

   (a) For every $s \in supp(lstate(\epsilon_1))$, $s.Src_{yval}.chosenval = \perp$.
   (b) For every $s \in supp(lstate(\epsilon_1))$, $s.Rec.yval = \perp$, and $lstate(\epsilon_1).Src_{yval}.chosenval$ is the uniform distribution on $\{0,1\} \rightarrow D$.
   (c) $lstate(\epsilon_1).Rec.yval$ is the uniform distribution on $\{0,1\} \rightarrow D$.

If (a) holds for ϵ_1 and u, then consider any $s' \in supp(lstate(\epsilon'_{1j}))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, a, \mu_s) \in D_{RS\|Env}$. We have by (a) that $s.Src_{yval}.chosenval = \bot$.

By definition of the effects of T, $s'.Src_{yval}.chosenval = s.Src_{yval}.chosenval = \bot$, and so (a) holds for ϵ'_{1j} and u'.

If (b) holds for ϵ_1 and u, then consider any $s' \in supp(lstate(\epsilon'_{1j}))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, a, \mu_s) \in D_{RS\|Env}$. We have by (b) that $s.Rec.yval\bot$. By the effects of T, $s'.Rec.yval = s.Rec.yval = \bot$, so the first part of (b) holds. For the second part of (b), recall that we have defined ϵ'_{1j} in such 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 do not affect the value of $Src_{yval}.chosenval$, we have that $lstate(\epsilon'_{1j}).Src_{yval}.chosenval = lstate(\epsilon_1).Src_{yval}.chosenval$, and so (b) holds for ϵ'_{1j} and u'.

If (c) holds for ϵ_1 and u, then we argue as for the second part of (b), using the fact that T transitions do not affect $Rec.yval$. Thus, (c) holds for ϵ'_{1j} and u'. Therefore, in all cases, Property 2 holds for ϵ'_{1j} and u', and hence for ϵ'_{1j} and ϵ'_{2j}.

Finally, we show that $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$. Since ϵ'_{1j} and ϵ'_{2j} are derived from ϵ'_1 and ϵ'_2 by $apply(,)$ and a is the unique action in T that is enabled in all states in $supp(\epsilon_1) \cup supp(\epsilon_2)$, we know that each trace in $supp(tdist(\epsilon'_{1j}))$ is of the form $\beta_1\ a$, where $\beta_1 \in supp(tdist(\epsilon_1))$, and each trace in $supp(tdist(\epsilon'_{2j}))$ is of the form $\beta_2\ a$, where $\beta_2 \in supp(tdist(\epsilon_2))$. In fact, $tdist(\epsilon'_{1j})(\beta_1\ a) = tdist(\epsilon_1)(\beta_1)$ and $tdist(\epsilon'_{2j})(\beta_2\ a) = tdist(\epsilon_2)(\beta_2)$. Since $tdist(\epsilon)_1 = tdist(\epsilon_2)$, we have $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$, as needed.

2. $T = \{in(i)_{Rec}\}$.

   Since T is an output task of Env, 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 $tr_{Env} = (q_{Env}, a, \mu_{Env})$ of Env from q_{Env} with action a. Here, $a = in(i)_{Rec}$ for a particular value of i.

   The rest of the proof for this case follows the proof for $T = \{in(x)_{Trans}\}$. 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 $Rec.inval$ and Env in the RS system, and $Funct.inval(Rec)$ and Env in the $Int1$ system, and use Properties 1(b) and 1(j) instead of Properties 1(a), 1(c), and 1(j).

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

   We first show that T is enabled in every state in $supp(lstate(\epsilon_2))$. Thus, fix any state $u \in supp(lstate(\epsilon_2))$; we show that T 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 Src_{tdpp}, T is enabled in $s.Src_{tdpp}$. The precondition of T in the definition of Src_{tdpp} implies that $s.Src_{tdpp}.chosenval = \bot$. By Property 1(g), $u.Src_{tdpp} = s.Src_{tdpp}$. So, T is enabled in $u.Src_{tdpp}$, and hence in u, as needed.

   Next 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 execution fragments $\alpha \in supp(\epsilon'_1)$ such that $lstate(\alpha).Src_{tdpp}.chosenval$ is the jth element in domain Tdp (in some enumeration). 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 Properties 1 and 2 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 1(g) holds for s' and u'. 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 proof for Property 2 is similar to the corresponding part of the proof for $T = \{in(x)_{Trans}\}$. For trace distribution equivalence, we must show that $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$. Since ϵ'_{1j} and ϵ'_{2j} are derived from ϵ'_1 and ϵ'_2 by $apply(,)$ and the actions that are enabled in states in $supp(\epsilon_1) \cup supp(\epsilon_2)$ are internal, $tdist(\epsilon_{1j}) = tdist(\epsilon_1)$ and $tdist(\epsilon_{2j}) = tdist(\epsilon_2)$. Since $tdist(\epsilon)_1 = tdist(\epsilon_2)$, we have $tdist(\epsilon'_{1j}) = tdist(\epsilon'_{2j})$, as needed.

4. $T = \{rand(p)_{tdpp}\}$.

   We first show that T is enabled in every state in $supp(lstate(\epsilon_2))$. Thus, fix any state $u \in supp(lstate(\epsilon_2))$; we show that T is enabled in u. Choose any $s \in supp(lstate(\epsilon_1))$. Since T is enabled in s and T is an output task of Src_{tdpp}, T is enabled in $s.Src_{tdpp}$ and so $s.Src_{tdpp}.chosenval \neq \perp$. By Property 1(g) for s and u, $u.Src_{tdpp} = s.Src_{tdpp}$. So, T is enabled in $u.Src_{tdpp}$, and hence in u, as needed.

   We show that there is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, as in the proofs for Claim 1 and Claim 2. Here, we use Property 1(g) instead of 1(j).

   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 $(s, a, \mu_s) \in D_{RS\|Env}$. 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 1(d) 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 1(d) 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 proofs for Property 2 and trace distribution equivalence are similar to the corresponding parts of the proof for $T = \{in(x)_{Trans}\}$, using ϵ'_1 and ϵ'_2 instead of ϵ'_{1j} and ϵ'_{2j}.

5. $T = \{choose - rand_{yval}\}$.

   Here, a random choice is made in the *RS* system but not in the *Int1* system.
   Since $corrtasks([lstate(\epsilon_1)], T) = \lambda$, no enabling condition needs to be shown. Also, we have $\epsilon'_2 = \epsilon_2$.

   Let p be the Dirac measure on the single index 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 $u' \in supp(lstate(\epsilon'_2))$. Since $\epsilon'_2 = \epsilon_2$, we know that $u' \in supp(lstate(\epsilon_2))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, choose - rand_{yval}, \mu_s) \in D_{RS\|Env}$. We know that Property 1 holds for s and u'. Observe that the application of T updates only the $s.Src_{yval}.chosenval$ component in the *RS* system and the application of λ leaves u' unchanged. Since Property 1 does not mention $Src_{yval}.chosenval$, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.

   To establish Property 2, consider any state $u' \in supp(lstate(\epsilon'_2))$ such that $u'.TR1.zval = \perp$. We show that Property 2(b) holds; that is, we show that for every $s' \in supp(lstate(\epsilon'_1))$, $s'.Rec.yval = \perp$, and $lstate(\epsilon'_1).Src_{yval}.chosenval$ is the uniform distribution on $\{0,1\} \rightarrow D$.

   Consider any $s' \in supp(lstate(\epsilon'_1))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, choose - rand_{yval}, \mu_s) \in D_{RS\|Env}$. Since $choose - rand_{yval}$ is enabled in s, we know that $s.Src_{yval}.chosenval = \perp$. Therefore, by Lemma 6.3, $s.Rec.yval = \perp$. Since T does not update *Rec.yval* we have $s'.Rec.yval = \perp$. Hence, the first part of 2(b) holds.

For the second part of 2(b), the effects of T imply that $Src_{yval}.chosenval$ is chosen according to the uniform probability distribution on domain $\{0,1\} \rightarrow D$. So, $lstate(\epsilon'_1).Src_{yval}.chosenval$ is the uniform distribution on $\{0,1\} \rightarrow D$, as needed.

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.

6. $T = \{rand(y)_{yval}\}$.

   Here, a step is taken in the RS system but not in the $Int1$ system. Since $corrtasks([lstate(\epsilon_1)], T) = \lambda$, no enabling condition needs to be shown, and $\epsilon'_2 = \epsilon_2$.

   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 $\epsilon'_{21} = \epsilon'_2$. 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.

   To establish Property 1, consider any states $s' \in supp(lstate(\epsilon'_1))$ and $u' \in supp(lstate(\epsilon'_2))$. Since $\epsilon'_2 = \epsilon_2$, we know that $u' \in supp(lstate(\epsilon_2))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, rand(y)_{yval}, \mu_s) \in D_{RS\|Env}$ and $y = (s.Src_{yval}.chosenval)$. We know that Property 1 holds for s and u'. Observe that the application of T updates only the $s.Rec.yval$ component in the RS system and the application of λ leaves u' unchanged. Since Property 1 does not mention $Rec.yval$, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.

   To establish Property 2, consider any state $u' \in supp(lstate(\epsilon'_2))$ such that $u'.TR1.zval = \perp$. We show that Property 2(c) holds; that is, we show that $lstate(\epsilon'_1).Rec.yval$ is the uniform distribution on $\{0,1\} \rightarrow D$.

   Since $u' \in supp(lstate(\epsilon_2))$, we know that Property 2 holds for u' and ϵ_1. However, 2(a) cannot hold because T is enabled in $supp(lstate(\epsilon_1))$, so either 2(b) or 2(c) must hold for u' and ϵ_1. If 2(b) holds for u' and ϵ_1, then consider any $s' \in supp(lstate(\epsilon'_1))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, rand(y)_{yval}, \mu_s) \in D_{RS\|Env}$ and $y = s.Src_{yval}.chosenval$. We know that $s.Rec.yval = \perp$ and $lstate(\epsilon_1).Src_{yval}.chosenval$ is the uniform distribution on $\{0,1\} \rightarrow D$. Then, by the effects of T and the definition of ϵ'_1, $s'.Rec.yval \neq \perp$ and $lstate(\epsilon'_1).Rec.yval$ is the uniform distribution on $\{0,1\} \rightarrow D$, and hence 2(c) holds for u' and ϵ'_1, as needed.

   On the other hand, if 2(c) holds for u' and ϵ_1, then we know that $lstate(\epsilon_1).Rec.yval$ is the uniform distribution on $\{0,1\} \rightarrow D$. Since the application of T affects $Rec.yval$ only if it is $\perp$, we know that $lstate(\epsilon'_1).Rec.yval = lstate(\epsilon_1).Rec.yval$. Therefore, in this case 2(c) holds for u' and ϵ'_1, as needed to show Property 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.

7. $T = \{fix-zval_{Rec}\}$.

   Here, a deterministic step in the RS system maps to a random choice in the $Int1$ system. We first show that the sequence of tasks $\{choose-rand_{zval}\}$ $\{rand_{zval}\}$ is enabled in $supp(lstate(\epsilon_2))$. First, consider any state $u \in supp(lstate(\epsilon_2))$; we show that $\{choose-rand_{zval}\}$ 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 Rec, T is enabled in $s.Rec$. By the precondition of the $fix-zval_{Rec}$ action in Rec, we know that $s.Rec.zval = \perp$. By Property 1(h) for s and u, $u.Src_{zval}.chosenval = \perp$. So, $\{choose-rand_{zval}\}$ is enabled in u, as needed.

   Now, let ϵ''_2 be the measure $apply(\epsilon_2, \{choose-rand_{zval}\})$. We claim that $\{rand(z)_{zval}\}$ is enabled in $supp(lstate(\epsilon''_2))$. Consider any state $u'' \in supp(lstate(\epsilon''_2))$. By the effect of $\{choose-rand_{zval}\}$, we know that $u''.Src_{zval}.chosenval \neq \perp$, which is the only precondition on actions in $\{rand(z)_{zval}\}$. Thus, $\{rand(z)_{zval}\}$ is enabled in $supp(lstate(\epsilon''_2))$, as needed.

   Next, we claim that $lstate(\epsilon_1).Rec.yval$ is the uniform distribution on $\{0,1\} \rightarrow D$. To see this, consider any pair of states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$. Since $s.Rec.zval = \perp$, by Property 1(e), we have $u.TR1.zval = \bot$. Then by Property 2 for u and ϵ_1, we know that one of the following holds:

(a) $s.Src_{yval}.chosenval = \bot$.

(b) $s.Rec.yval = \bot$ and $lstate(\epsilon_1).Src_{yval}.chosenval$ is the uniform distribution on $\{0,1\} \to D$.

(c) $lstate(\epsilon_1).Rec.yval$ is the uniform distribution on $\{0,1\} \to D$.

However, since T is enabled in $supp(lstate(\epsilon_1))$, we know that $s.Rec.yval \neq \bot$, so (b) cannot hold. Using Lemma 6.3, we see that also (a) cannot hold. Therefore, (c) holds; that is, $lstate(\epsilon_1).Rec.yval$ is the uniform distribution on $\{0,1\} \to D$, as needed.

Next, we show that $lstate(\epsilon'_1).Rec.zval$ is the uniform distribution on $\{0,1\} \to D$: By Property 1(b), $Rec.inval$ is the same in all states in $supp(lstate(\epsilon_1))$. By Lemma 6.4 5(b) and Property 1(d), $Rec.tdp$ is the same in every state in $supp(lstate(\epsilon_1))$. The effect of a $fix-zval_{Rec}$ action gives $Rec.zval(inval) = tdp(yval(inval))$ and $Rec.zval(1-inval) = yval(1-inval)$ where tdp is a permutation. Thus, since $lstate(\epsilon_1).Rec.yval$ is the uniform distribution on $\{0,1\} \to D$, it follows that $lstate(\epsilon'_1).Rec.zval$ is also the uniform distribution on $\{0,1\} \to D$.

Next, 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 $r = |\{0,1\} \to D| = |D|^2$. 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 execution fragments $\alpha \in supp(\epsilon'_1)$ such that $lstate(\alpha).Rec.zval$ is the jth element of the domain $\{0,1\} \to D$. For each $\alpha \in supp(\epsilon'_{1j})$ of the form $\alpha'\, fix-zval_{Rec}\, q$, let $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$. Similarly, we define probability measure ϵ'_{2j} as follows. The support $supp(\epsilon'_{2j})$ is the set of execution fragments $\alpha \in supp(\epsilon'_2)$ such that $lstate(\alpha).TR1.zval$ is the jth element of the domain $\{0,1\} \to D$. For each $\alpha \in supp(\epsilon'_{2j})$ of the form $\alpha'\, choose-rand_{zval}\, q\, rand_{zval} q'$, let $\epsilon'_{2j}(\alpha) = \epsilon_2(\alpha')$.

Now fix $j \in I$; we show that $(\epsilon'_{1j}, \epsilon'_{2j}) \in R$. To do this, we establish Properties 1 and 2 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 RS and $Int1$, we know that application of T updates $Rec.zval$ in the RS system and application of the sequence $\{choose-rand_{zval}\}\ \{rand_{zval}\}$ updates $Src_{zval}.chosenval$ and $TR1.zval$ in the $Int1$ system. We show that Properties 1(e) and 1(h) hold for u' and s'.

Property 1(e) follows from the definitions of ϵ'_{1j} and ϵ'_{2j}; both actions give the same element of the domain $\{0,1\} \to D$ when projected onto $Rec.zval$ and $TR1.zval$. For Property 1(h), we use the fact that $u'.TR1.zval = s'.Rec.zval$, and we observe in addition that if $u'.TR1.zval \neq \bot$, then $u'.TR1.zval = u'.Src_{zval}.chosenval$, by Lemma 9.3. Since no state component other than $Rec.zval$ in the RS system is updated by the application of T, and no state component other than $TR1.zval(Trans)$ and $Src_{zval}.chosenval$ is updated by the application of $\{choose-rand_{zval}\}\ \{rand_{zval}\}$ in the $Int1$ system, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.

Property 2 holds trivially in this case since for any state $u' \in supp(lstate(\epsilon'_{2j}))$, we have $u'.TR1.zval \neq \bot$ by definition of ϵ'_{2j}.

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}.

8. $T = \{fix-bval_{Trans}\}$

We first show that T is enabled in every state in $supp(lstate(\epsilon_2))$. Fix any state $u \in supp(lstate(\epsilon_2))$; we show that T 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 $s.Trans$, and $s.Trans.zval \neq \bot$, $s.Trans.tdpp \neq \bot$, $s.Trans.inval \neq \bot$, and $s.Trans.bval = \bot$. By Property 1(c), $u.TR1.inval(Trans) = s.Trans.inval \neq \bot$. By Property 1(d), $u.TR1.tdpp = s.Trans.tdpp \neq \bot$. By Lemma 6.4 7(b), $s.Rec.zval \neq \bot$, and by Property 1(e), $u.TR1.zval = s.Rec.zval \neq \bot$. Finally, by Property 1(f), $u.TR1.bval = \bot$. So, T is enabled in $u.TR1$, and hence in u, as needed.

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 $(s, fix - bval_{Trans}, \mu_s) \in D_{RS\|Env}$. Similarly, let u be any state in $supp(lstate(\epsilon_2))$ such that $u' \in supp(\mu_u)$, where $(u, fix-bval_{Trans}, \mu_u) \in D_{Int1\|Env}$.

By definitions of RS and $Int1$ we know that application of T updates $Trans.bval$ in the RS system and $TR1.bval$ in the $Int1$ system. By the effects of T in the two systems, we know that $u'.TR1.bval = s'.Trans.bval$; hence, Property 1(f) holds. Since no state component other than $Trans.bval$ in the RS system, and $TR1.bval$ 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.

To establish Property 2, consider any state $u' \in supp(lstate(\epsilon'_2))$. We show that $u'.TR1.zval \neq \bot$, and therefore Property 2 of R holds trivially. Let u be some state in $supp(lstate(\epsilon_2))$ such that $u' \in supp(\mu_u)$ where $(u, fix - bval_{Trans}, \mu_u) \in D_{Int1\|Env}$. Since T is enabled in u, we know that $u.TR1.zval \neq \bot$. By the effects of T, we know that $u'.TR1.zval = u.TR1.zval \neq \bot$, as claimed.

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.

9. $T = \{send(1, f)_{Trans}\}$.

   We first show that T is enabled in every state in $supp(lstate(\epsilon_2))$. Fix any state $u \in supp(lstate(\epsilon_2))$; we show that T is enabled in u. Choose any $s \in supp(lstate(\epsilon_1))$. Since T is enabled in s and T is an output task of $Trans$, T is enabled in $s.Trans$, and so $s.Trans.tdpp \neq \bot$. By Property 1(d), $u.TR1.tdpp = s.Trans.tdpp$. So, T is enabled in $u.TR1$, and hence in u, as needed.

   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 1(d) 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 to refer to $send(1, Trans.tdpp.funct)$ 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 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 $(s, a, \mu_s) \in D_{RS\|Env}$. 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 both the RS and $Int1$ systems. By Property 1(i), $u.Adv = s.Adv$. It is obvious that $u'.Adv = s'.Adv$ and that 1(i) holds, since Adv is the same automaton in both systems. Since no component other than $Adv.messages$ is updated, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.

   The proofs for Property 2 and trace distribution equivalence are similar to the corresponding parts of the proof for $T = \{in(x)_{Trans}\}$, using ϵ'_1 and ϵ'_2 instead of ϵ'_{1j} and ϵ'_{2j}.

10. $T = \{send(2, z)_{Rec}\}$.

    We first show that T is enabled in every state in $supp(lstate(\epsilon_2))$. Fix any state $u \in supp(lstate(\epsilon_2))$; we show that T is enabled in u. Choose any $s \in supp(lstate(\epsilon_1))$. Since T is enabled in s and T

is an output task of *Rec*, T is enabled in $s.Rec$, and therefore $s.Rec.zval \neq \bot$. By Property 1(e), $u.TR1.zval = s.Rec.zval \neq \bot$. So, T is enabled in $u.Rec$, and hence in u, as needed.

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 1(e) 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 $T = \{send(1, f)_{Trans}\}$.

11. $T = \{send(3, b)_{Trans}\}$.

    The proof that T is enabled in every state in $supp(lstate(\epsilon_2))$ is analogous to the corresponding part of the proof for $T = \{send(1, f)_{Trans}\}$, using Property 1(f) instead of 1(d).

    We also 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 $T = \{send(1, f)_{Trans}\}$.

12. $T = \{receive(1, f)_{Rec}\}$.

    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, a is $receive(1, f)_{Rec}$ for a fixed value of f.

    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 use the fact that application of T updates only $Rec.tdp$ in RS and that R does not depend on this component.

13. $T = \{receive(2, z)_{Trans}\}$.

    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 a is $receive(2, z)_{Trans}$ for a fixed value of z.

    The rest of the proof differs from the proof for $T = \{receive(1, f)_{Rec}\}$ only in showing that Property 1 holds; here we use the fact that the application of T updates $Trans.zval$ only, which has no effect R.

14. $T = \{receive(3, b)_{Rec}\}$.

    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 a is $receive(3, b)_{Rec}$ for a fixed value of b.

    The rest of the proof differs from the porof for $T = \{receive(1, f)_{Rec}\}$ only in showing that Property 1 holds; here, we use the fact that the application of T updates $Rec.outval$ only, which has no effect on R.

15. $T = \{out(x)_{Rec}\}$.

    We first show that T is enabled in every state in $supp(lstate(\epsilon_2))$. So, fix any state $u \in supp(lstate(\epsilon_2))$; we show that T is enabled in u. Note that T is an output task of *Funct* in the *Int1* system. Choose any $s \in supp(lstate(\epsilon_1))$. Since T is enabled in s and T is an output task of *Rec* in *RS*, T is enabled in $s.Rec$ and $s.Rec.outval \neq \bot$. Then, by Lemma 6.4 9(b), we know that $s.Rec.outval = s.Trans.inval(s.Rec.inval) \neq \bot$. By Property 1(a), $u.Funct.inval(Trans) = s.Trans.inval$ and by Property 1(b) $u.Funct.inval(Rec) = s.Rec.inval$. Therefore, we have that $u.Funct.inval(Trans) \neq \bot$ and $u.Funct.inval(Rec) \neq \bot$. So, T is enabled in $u.Funct$, and hence in u, as needed.

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 1(a) that $Trans.inval$ is the same in all states in $supp(lstate(\epsilon_1))$ and by Property 1(b) that $Rec.inval$ is the same in all states in $supp(lstate(\epsilon_1))$. Since T is enabled in $supp(lstate(\epsilon_1))$, we know by the precondition of actions in T and by Lemma 6.4 9(b) that $out(s.Trans.inval(s.Rec.inval))$ is the unique action in T that is enabled in $supp(lstate(\epsilon_1))$. We use a to refer to $out(s.Trans.inval(s.Rec.inval))$ in the rest of the proof for this case. Similarly, by Property 1(a) we know that $Funct.inval(Trans)$ is the same in all states in $supp(lstate(\epsilon_2))$ and is equal to $Trans.inval$. By Properties 1(b) we know that $Funct.inval(Rec)$ is the same in all states in $supp(lstate(\epsilon_2))$ and is equal to $Rec.inval$. Hence, a is also the unique action that is enabled in $supp(lstate(\epsilon_2))$, and thus in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$, as needed.

Then next-transition determinism for Env implies that there is a unique transition of Env from q_{Env} with action a. Let $tr_{Env} = (q_{Env}, a, \mu_{Env})$ be this unique transition.

Next 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 execution fragments $\alpha \in supp(\epsilon'_1)$ 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 $j \in I$; we show that $(\epsilon'_{1j}, \epsilon'_{2j}) \in R$. To do this, we establish Properties 1 and 2 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 $s' \in supp(lstate(\epsilon'_1 j))$ and $u' \in supp(lstate(\epsilon'_2 j))$. 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 the definitions of the RS and $Int1$ systems, we know that application of T does not update any state component of RS or $Int1$; however, it may update the state of Env in both systems. Since Property 1 holds for s and u, we know that all the parts of Property 1 except possible for 1(j) also hold for s' and u'. We also know that 1(j) holds for s' and u' by definition of ϵ'_{1j} and ϵ'_{2j}: in both s' and u', the state of Env is q_j. Thus, Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.

The proofs for Property 2 and trace distribution equivalence are similar to the corresponding parts of the proof for $T = \{in(x)_{Trans}\}$.

16. T is an output task of Env and an input task of Adv.

    Since T is an output task of Env, 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 $tr_{Env} = (q_{Env}, a, \mu_{Env})$ of Env from q_{Env} with action a. Also, by next-transition determinism, it follows 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 execution fragments $\alpha \in supp(\epsilon'_1)$ 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 $T = \{in(x)_{Trans}\}$. 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* (by definition of the *RS* and *Int1* systems) and use Properties 1(i) and 1(j).

17. T is either an output task of *Env* that is not an input task of *Adv*, *Trans*, or *Rec*, or is an internal task of *Env*.

    Since T is an output or internal task of *Env*, 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 $tr_{Env} = (q_{Env}, a, \mu_{Env})$ of *Env* from q_{Env} with action a.

    To show the step correspondence, we proceed as for $T = \{in(x)_{Trans}\}$. 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 1(j).

18. T is an output task of *Adv* and an input task of *Env*.

    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. 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 $T = \{in(x)_{Trans}\}$, using Properties 1(i) and 1(j).

    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}.

19. T is either an output task of *Adv* that is not an input task of *Env*, *Trans*, or *Rec*, or is an internal task of *Adv*.

    Since T is an output or internal 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.

    To show the step correspondence, we proceed as for $T = \{in(x)_{Trans}\}$, but using *Adv* instead of *Env*. In showing that Property 1 holds 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 1(i).

    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}.

$\square$

Proof. (Of Lemma 9.5:)
By Lemma 9.7, R is a simulation relation from $RS_k \| Env$ to $Int1_k \| Env$. Then Theorem 3.53 implies that $tdists(RS_k \| Env) \subseteq tdists(Int1_k \| Env)$. Since *Env* was chosen arbitrarily, this implies (by definition of $\leq_0$) that $RS_k \leq_0 Int1_k$. $\square$

Proof. (Of Lemma 9.6:)
By Lemma 9.7, R is a simulation relation from $RS_k \| Env$ to $Int1_k \| Env$ for which $|corrtasks(S, T)| \leq 2$ for every S and T. Also, note that Lemma 9.7 holds for every k and for every environment *Env* for *RS* and *Int1* (without any time-bound assumption). Thus, the hypotheses of Theorem 3.86 are satisfied, so by that theorem, $\overline{RS} \leq_{neg,pt} \overline{Int1}$. $\square$

9.5 *Int1* implements *Int2*

This step introduces an ϵ-approximation into the implementation relation, for some negligible function ϵ that is obtained from the definition of a hard-core predicate. We show:

Lemma 9.8 *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:

- $\overline{SInt1} = hide_{\{rand(*)_{tdpp}\} \cup \{rand(*)_{zval}\}}(\overline{TR1} \| \overline{Src_{tdpp}} \| \overline{Src_{zval}})$,
- $\overline{SInt2} = hide_{\{rand(*)_{tdpp}\} \cup \{rand(*)_{zval}\} \cup \{rand(*)_{cval}\}}(\overline{TR2} \| \overline{Src_{tdpp}} \| \overline{Src_{zval}} \| \overline{Src_{cval}})$.

Next, using mappings of the sort we used in Section 9.4, we will show that $\overline{SInt1} \leq_0 \overline{SHOT}$ and $\overline{SHROT} \leq_0 \overline{SInt2}$ or, more precisely, that $SInt1_k \leq_0 SHOT_k$ and $SHROT_k \leq_0 SInt2_k$ for every k. In the rest of this subsection, we suppress the mention of k everywhere.

Finally, using the properties of these mappings and the different properties of the $\leq_{neg,pt}$ relation, we will prove the expected relation.

9.5.1 The *SInt1* subsystem implements *SHOT*

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 execution fragments 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:

1. For every $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$:
   (a) $u.Ifc.xval = s.TR1.inval(Trans)$.
   (b) if $s.Src_{tdpp}.chosenval \neq \perp$ then $u.Src_{tdp}.chosenval = s.Src_{tdpp}.chosenval.funct$.
   (c) $u.Src_{yval0}.chosenval = \perp$ iff $s.Src_{zval}.chosenval = \perp$.
   (d) $u.Src_{yval1}.chosenval = \perp$ iff $s.Src_{zval}.chosenval = \perp$.
   (e) if $s.Src_{zval}.chosenval \neq \perp$ then
   $u.H0.yval = s.Src_{tdpp}.chosenval.inverse(s.Src_{zval}.chosenval(0))$ and
   $u.H1.yval = s.Src_{tdpp}.chosenval.inverse(s.Src_{zval}.chosenval(1))$
   (f) $u.H0.zval = s.Src_{zval}.chosenval(0)$ and $u.H1.zval = s.Src_{zval}.chosenval(1)$.
   (g) if $s.TR1.tdpp \neq \perp$ then $u.Ifc.fval = s.TR1.tdpp.funct$.
   (h) $u.Ifc.zval = s.TR1.zval$.
   (i) If $u.Ifc.bval \neq \perp$ then $u.Ifc.bval = B(s.TR1.tdpp.inverse(s.TR1.zval))$.
   (j) $u.Ifc.bxorx = s.TR1.bval$.
   (k) $u.Env' = s.Env'$.
2. For every $s \in support(lstate(\epsilon_1))$:
   If $s.Src_{tdpp}.chosenval = \perp$ then one of the following holds:
   (a) s.$Src_{zval}.chosenval = \perp$ and, for every $u \in support(lstate(\epsilon_2))$, $u.Src_{tdp}.chosenval = \perp$. (That is, f has not yet been chosen in $SHOT$.)
   (b) $s.Src_{zval}.chosenval \neq \perp$ and, for every $u \in support(lstate(\epsilon_2))$, $u.Src_{tdp}.chosenval \neq \perp$, and $u.Ifc.fval = \perp$; also $lstate(\epsilon_2).Src_{tdp}.chosenval$ is the uniform distribution on Tdp. (That is, the choice has already been made in $SHOT$, but has not yet been communicated by Src_{tdp} to $H0$, $H1$, and Ifc.)

(c) $s.Src_{zval}.chosenval \neq \bot$ and $lstate(\epsilon_2).Ifc.fval$ is the uniform distribution on Tdp. (That is, the choice has already been made in $SHOT$, and communicated to the other components.)

Lemma 9.9 *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 eight steps of $SHOT\|Env'$, that is, for every S, T, $|corrtasks(S,T)| \leq 8$.*

there are several possible step correspondences, and we are not sure that we exhibit the most efficient one. Maybe we could formulate these lemmas as:

Proof. We prove that R satisfies the two conditions in Lemma 3.55.
Start condition: It is obvious that the Dirac measures on execution fragments consisting of the unique start states s and u of $SInt1\|Env'$ and $SHOT\|Env'$, respectively, are R-related. Property 1 of R holds because the state components of s and u on which R depends are all $\bot$. Property 2 of R holds because $s.Src_{zval}.chosenval = \bot$ and $u.Src_{tdp}.chosenval = \bot$.
Step condition: We define $corrtasks(RS_{SInt1\|Env'} \times RA_{SInt1\|Env'}) \rightarrow RA^*_{SHOT\|Env'}$ as follows:
For any $(S,T) \in (RS_{SInt1\|Env'} \times RA_{SInt1\|Env'})$:

- If $T = \{in(x)_{Trans}\}$ then $corrtasks(S,T) = \{in(x)_{Trans}\}$.
- If $T = \{choose{-}rand_{tdpp}\}$ and $s.Src_{zval}.chosenval = \bot$ in every state s of S then $corrtasks(S,T) = \{choose - rand_{tdp}\}$.
- If $T = \{choose{-}rand_{tdpp}\}$ and $s.Src_{zval}.chosenval \neq \bot$ in every state s of S then $corrtasks(S,T) = \lambda$.
- If $T = \{choose - rand_{zval}\}$ and $s.TR1.tdpp \neq \bot$ in every state s of S then $corrtasks(S,T) = \{choose{-}rand_{yval0}\}\{choose{-}rand_{yval1}\}\{rand(y)_{yval0}\}\{rand(y)_{yval1}\}\{fix{-}zval_0\}\{fix{-}zval_1\}$.
- If $T = \{choose - rand_{zval}\}$ and $s.TR1.tdpp = \bot$ and $s.Src_{tdpp}.chosenval \neq \bot$ in every state s of S then $corrtasks(S,T) = \{rand(f)_{tdp}\}\{choose - rand_{yval0}\}\{choose - rand_{yval1}\}\{rand(y)_{yval0}\}$ $\{rand(y)_{yval1}\}\{fix - zval_0\}\{fix - zval_1\}$. This case corresponds to the fact that the random permutation has already been chosen in $SHOT$, but not yet transmitted to $H0$ and $H1$.
- If $T = \{choose{-}rand_{zval}\}$ and $s.Src_{tdpp}.chosenval = \bot$ in every state s of S then $corrtasks(S,T) = \{choose - rand_{tdp}\}\{rand(f)_{tdp}\}\{choose - rand_{yval0}\}\{choose - rand_{yval1}\}\{rand(y)_{yval0}\}$ $\{rand(y)_{yval1}\}\{fix - zval_0\}\{fix - zval_1\}$. This case corresponds to the fact that the random permutation has not yet been chosen in $SHOT$.
- If $T = \{rand(p)_{tdpp}\}$ then $corrtasks(S,T) = \{rand(f)_{tdp}\}$.
- If $T = \{rand(z)_{zval}\}$ then $corrtasks(S,T) = \{rand(z)_{zval0}\}\{rand(z)_{zval1}\}$.
- If $T = \{fix - bval_{Trans}\}$ then $corrtasks(S,T) = \{fix - bval_0\}\{fix - bval_1\}\{rand(b)_{bval0}\}$ $\{rand(b)_{bval1}\}\{fix - bxorx\}$.
- If $T \in \{\{send(1,f)_{Trans}\}, \{send(2,z)_{Rec}\}, \{send(3,b)_{Trans}\}\}$ then $corrtasks(S,T) = \{T\}$.

There are two interesting points in this correspondence. The first one comes from the fact that the z-values are chosen randomly in *Int1* while they are computed as the image of randomly selected y-values through the permutation f in $SHOT$. This difference imposes that, in $SHOT$, the trapdoor permutation f must have been selected in order to be able to compute the z-values.

The second interesting point comes from the fact that the b-values are computed as $B(f^{-1}(z))$ in *SInt1* and as $B(y)$ in $SHOT$. As a consequence of this, f must have been selected in order to compute the b-values in *SInt1*, while this is not necessary in $SHOT$. However, this does not require any specific treatment here as the *corrtasks* function is only applied on enabled tasks: it is therefore not possible that $SHOT$ performs a $fix - bval_{Trans}$ step without any corresponding step of *Int1*.

Suppose $(\epsilon_1, \epsilon_2) \in R$ and T is a task of $SInt1\|Env'$ that is enabled in $supp(lstate(\epsilon_1))$. 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 9.7. State equivalence follows as in that proof. 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 (and S when needed).

1. $T = \{in(x)_{Trans}\}$ then $corrtasks(S, T) = \{in(x)_{Trans}\}$

   The treatment of this case is similar as the one described in the proof of Lemma 9.7.

2. $T = \{choose - rand_{tdpp}\}$ and $s.Src_{zval}.chosenval = \perp$ in every state s of S.

   We first show that $T' = corrtasks(S, T) = \{choose - rand_{tdp}\}$ is enabled in every state in $supp(lstate(\epsilon_2))$. Thus, fix any state $u \in supp(lstate(\epsilon_2))$; we show that T' 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 Src_{tdpp}, T is enabled in $s.Src_{tdpp}$. The precondition of T in the definition of Src_{tdpp} implies that $s.Src_{tdpp}.chosenval = \perp$. Now, since $Src_{zval}.chosenval = \perp$ and $\epsilon_1 R \epsilon_2$, we know that $s.Src_{tdp}.chosenval = \perp$.

   Next, we define the probability measures needed to show the 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 the probability measure ϵ_{1j}' as follows. The support $supp(\epsilon_{1j}')$ is the set of execution fragments $\alpha \in supp(\epsilon_1')$ such that $lstate(\alpha).Src_{tdpp}.chosenval$ is the j-th element in domain $Tdpp$ (in some enumeration). 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', assuming that the enumeration of the elements of the domain of Tdp is performed in the same order as the enumeration of the permutation pairs of $Tdpp$, that is, the j-th permutation of Tdp is also the j-th permutation of $Tdpp$.

   Now, it is easy to check that $\epsilon_{1j}' R \epsilon_{2j}'$: for any states $s' \in supp(lstate(\epsilon_{1j}')$ and $u' \in supp(lstate(\epsilon_{2j}')$ the only updated components are $s'.Src_{tdpp}.chosenval$ and $u'.Src_{tdp}.chosenval$, they are different from $\perp$ and $s'.Src_{tdpp}.chosenval.funct = u'.Src_{tdp}.chosenval$.

3. $T = \{choose - rand_{tdpp}\}$ and $s.Src_{zval}.chosenval \neq \perp$ in every state s of S.

   Since, in that case, $corrtasks([lstate(\epsilon_1)], T) = \lambda$, no enabling condition needs to be shown and $\epsilon_2' = \epsilon_2$.

   Next, we define the probability measures needed to show the 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 the probability measure ϵ_{1j}' as follows. The support $supp(\epsilon_{1j}')$ is the set of execution fragments $\alpha \in supp(\epsilon_1')$ such that $lstate(\alpha).Src_{tdpp}.chosenval$ is the j-th element in domain $Tdpp$ (in some enumeration). For each $\alpha \in supp(\epsilon_{1j}')$ of the form $\alpha' choose - rand_{tdpp} q$, let $\epsilon_{1j}'(\alpha) = \epsilon_1(\alpha')$.

   Now, we define ϵ_{2j}' from ϵ_2'. The support $supp(\epsilon_{2j}')$ is the set of execution fragments $\alpha \in supp(\epsilon_2')$ such that $lstate(\alpha).Src_{tdp}.chosenval$ is the j-th element in domain Tdp (according to the same enumeration as above). Furthermore, for each $\alpha \in supp(\epsilon_{2j}')$, let $\epsilon_{2j}'(\alpha) = \epsilon_2'(\alpha)$ (this is acceptable since $lstate(\epsilon_2').Src_{tdp}.chosenval$ is the uniform distribution on Tdp).

   Now, it is easy to check that $\epsilon_{1j}' R \epsilon_{2j}'$: for any states $s' \in supp(lstate(\epsilon_{1j}')$ and $u' \in supp(lstate(\epsilon_{2j}')$ the only updated component is $s'.Src_{tdpp}.chosenval$, which becomes different from $\perp$, and $s'.Src_{tdpp}.chosenval.funct = u'.Src_{tdp}.chosenval$.

4. $T = \{choose - rand_{zval}\}$ and $s.TR1.fval \neq \perp$ in every state s of S.

   We first check that all tasks in the sequence $corrtasks(S, T) = \{choose - rand_{yval0}\}\{choose - rand_{yval1}\}\{rand(y)_{yval0}\}$ $\{rand(y)_{yval1}\}\{fix - zval_0\}\{fix - zval_1\}$ are enabled. Thus, fix any state $u \in supp(lstate(\epsilon_2))$; we show that the sequence of tasks $corrtasks(S, T)$ is enabled in u. Choose any $s \in supp(lstate(\epsilon_1))$.

The $\{choose - rand_{yval0}\}$ and $\{choose - rand_{yval1}\}$ tasks are enabled because R guarantees that $u.Src_{yval0}.chosenval = \bot$ and $u.Src_{yval1}.chosenval = \bot$ when $s.Src_{zval}.chosenval = \bot$, which is the case since $T = \{choose - rand_{zval}\}$ is enabled.

Next, $\{rand(y)_{yval0}\}$ and $\{rand(y)_{yval1}\}$ are enabled because $u.Src_{yval0}.chosenval \neq \bot$ and $u.Src_{yval1}.chosenval \neq \bot$ now. Finally, the $\{fix - zval_0\}$ and $\{fix - zval_1\}$ tasks are enabled because

- $u.H0.fval \neq \bot$ and $u.H1.fval \neq \bot$, which is guaranteed by the assumption that $s.TR1.fval \neq \bot$
- $u.H0.yval \neq \bot$ and $u.H1.yval \neq \bot$, which is guaranteed by the execution of the $\{rand(y)_{yval0}\}$ and $\{rand(y)_{yval1}\}$ tasks just before.

Next, we define the probability measures needed to show the correspondence. Let p be the uniform probability measure on the index set $I = \{1 \cdots r\}$ where $r = |D|^2$; that is, $p(j) = 1/r$ for each $j \in I$.

The support $supp(\epsilon'_{1j})$ is the set of execution fragments $\alpha \in supp(\epsilon'_1)$ such that $lstate(\alpha).Src_{zval}.chosenval$ is the j-th element in domain $\{0,1\} \to D$ (in some enumeration). For each $\alpha \in supp(\epsilon'_{1j})$ of the form $\alpha' choose - rand_{zval} q$, let $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$.

Now, we define ϵ'_{2j} from ϵ'_2. The support $supp(\epsilon'_{2j})$ is the set of execution fragments $\alpha \in supp(\epsilon'_2)$ such that $(lstate(\alpha).H0.zval, lstate(\alpha).H1.zval) = (z(0), z(1))$ where z is the j-th element in domain $\{0,1\} \to D$ (according to the same enumeration as above). This correspondence preserves the trace distributions since $u.Src_{yval0}.chosenval$ and $u.Src_{yval1}.chosenval$ are selected from D according to the uniform distribution and $u.H0.zval$ and $u.H1.zval$ are computed as the image of these two elements of D through a permutation.

Now, it is easy to check that $\epsilon'_{1j} R \epsilon'_{2j}$: for any states $s' \in supp(lstate(\epsilon'_{1j}))$ and $u' \in supp(lstate(\epsilon'_{2j}))$ the only updated components are

- $s'.Src_{zval}.chosenval$, $u'.Src_{yval0}.chosenval$ and $u'.Src_{yval1}.chosenval$ which all become different from $\bot$, and
- $u'.H0.zval$ and $u'.H1.zval$ which remain equal to $s'.Src_{zval}.chosenval(0)$ and $s'.Src_{zval}.chosenval(1)$.

5. $T = \{choose - rand_{zval}\}$ and $s.TR1.tdpp = \bot$ and $s.Src_{tdpp}.chosenval \neq \bot$ in every state s of S.

   $corrtasks(S,T)$ is defined in the same way as in the previous case, except that we add a new task at the beginning of $corrtasks(S,T)$: $\{rand(f)_{tdp}\}$.

   This task is enabled since we now that $s.Src_{tdpp}.chosenval \neq \bot$ in every state s of S. Now, we can define the probability measures needed to show the correspondence in a similar way as in the previous case. The state variables which are changed in this case are those considered in the previous case, except $H0.fval$, $H1.fval$ and $Ifc.fval$ which were equal to $\bot$ in all states of ϵ_2 and become the uniform distribution on Tdp in ϵ'_2.

6. $T = \{choose - rand_{zval}\}$ and $s.Src_{tdpp}.chosenval = \bot$ in every state s of S.

   $corrtasks(S,T)$ is defined in the same way as in the previous case, except that we add a new task at the beginning of $corrtasks(S,T)$: $\{choose - rand_{tdp}\}$.

   This task is enabled since we now that $s.Src_{tdpp}.chosenval = \bot$ in every state s of S. Furthermore, executing $\{choose - rand_{tdp}\}$ enables the $\{rand(f)_{tdp}\}$ task. The other tasks are enabled for the same reasons as above.

   Now, we can define the probability measures needed to show the correspondence in a similar way as in the previous case. The state variables which are changed in this case are those considered in the previous case, except $Src_{tdp}.chosenval$ which was equal to $\bot$ in all states of ϵ_2 and become the uniform distribution on Tdp in ϵ'_2.

7. $T = \{rand(p)_{tdpp}\}$.

   The treatment of this case is similar to the corresponding one in the proof of Lemma 9.7.

8. $T = \{rand(z)_{zval}\}$.

   The treatment of this case is similar to the corresponding one in the proof of Lemma 9.7.

9. $T = \{fix - bval_{Trans}\}$.

   We first check that all tasks in the sequence $corrtasks(S,T) = \{fix - bval_0\}\{fix - bval_1\}$ $\{rand(b)_{bval0}\}\{rand(b)_{bval1}\}\{fix - bxorx\}$ are enabled. Thus, fix any state $u \in supp(lstate(\epsilon_2))$; we show that the sequence of tasks $corrtasks(S,T)$ is enabled in u. Choose any $s \in supp(lstate(\epsilon_1))$.

   The fact that T is enabled in s guarantees that $s.TR1.tdpp \neq \perp$, $s.TR1.zval \neq \perp$, $s.TR1.inval(Trans) \neq \perp$ and $s.TR1.bval = \perp$. We then conclude that:

   - the $\{fix{-}bval_0\}$ and $\{fix{-}bval_1\}$ tasks are enabled since they only require that $u.H0.yval \neq \perp$ and $u.H1.yval \neq \perp$, which is guaranteed by the fact that $s.TR1.zval \neq \perp$,
   - the $\{rand(b)_{bval0}\}$ and $\{rand(b)_{bval1}\}$ tasks are enabled since they only require $u.H0.bval \neq \perp$ and $u.H1.bval \neq \perp$, which is guaranteed by the fact that we just executed the the $\{fix - bval_0\}$ and $\{fix - bval_1\}$ tasks.
   - the $\{fix - bxorx\}$ is enabled since it requires that:
     - $u.Ifc.bval(i) \neq \perp$ $(i \in \{0,1\}$, which is guaranteed by the fact that we just executed the $\{rand(b)_{bval0}\}$ and $\{rand(b)_{bval1}\}$ tasks,
     - $u.Ifc.xval \neq \perp$, which is guaranteed by the fact that $s.TR1.inval(Trans) \neq \perp$ and the relation R,
     - $u.Ifc.fval \neq \perp$, which is guaranteed by the fact that $s.TR1.tdpp \neq \perp$ and the relation R,
     - $u.Ifc.zval \neq \perp$, which is guaranteed by the fact that $s.TR1.zval \neq \perp$ and the relation R,
     - $u.Ifc.bxorx = \perp$, which is guaranteed by the fact that $s.TR1.bval = \perp$ and the relation R.

   $s.TR1.bval(i)$ and $u.Ifc.bxorx$ compute the same values in $SInt1$ and $SHOT$, so the remaining part of the mapping do not raise any specific problem.

10. $T \in \{\{send(1,f)_{Trans}\}, \{send(2,z)_{Rec}\}, \{send(3,b)_{Trans}\}\}$

    The treatment of these cases is similar to the corresponding ones in the proof of Lemma 9.7.

□

9.5.2 $SHROT$ implements the $Int2$ 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 execution fragments 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:

1. For every $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$:

   (a) $s.Ifc.xval = u.TR2.inval(Trans)$.

   (b) $s.Src_{tdp}.chosenval = u.Src_{tdpp}.chosenval.funct$.

   (c) $s.Ifc.fval = u.TR2.tdpp.funct$.

(d) $u.Src_{zval}.chosenval \neq \bot$ iff for $i \in \{0,1\}$, $s.Src_{zvali} \neq \bot$.

(e) If $u.Src_{zval}.chosenval \neq \bot$ then for $i \in \{0,1\}$, $u.Src_{zval}.chosenval(i) = s.Src_{zvali}$.

(f) $u.Src_{cval}.chosenval \neq \bot$ iff for $i \in \{0,1\}$, $s.Src_{bvali} \neq \bot$.

(g) If $u.Src_{cval}.chosenval \neq \bot$ then for $i \in \{0,1\}$, $u.Src_{cval}.chosenval(i) = s.Src_{bvali}$.

(h) $u.TR2.zval \neq \bot$ iff for $i \in \{0,1\}$, $s.Ifc.zval(i) \neq \bot$.

(i) If $u.TR2.zval \neq \bot$ then $u.TR2.zval = s.Ifc.zval$.

(j) $u.TR2.cval \neq \bot$ iff for $i \in \{0,1\}$, $s.Ifc.bval(i) \neq \bot$.

(k) If $u.TR2.cval \neq \bot$ then $u.TR2.cval = s.Ifc.bval$.

(l) $s.Ifc.bxorx = u.TR2.bval$.

(m) $u.Env' = s.Env'$.

2. For every $u \in supp(lstate(\epsilon_1))$:

(a) If $u.Src_{zval}.chosenval = \bot$ then one of the following holds:

i. For every $s \in supp(lstate(\epsilon_1))$, for $i \in \{0,1\}$, $s.Src_{zvali} = \bot$.

ii. For some $i \in \{0,1\}$: for every $s \in supp(lstate(\epsilon_1))$, $s.Src_{zvali} = \bot$, and $s.Src_{zval(1-i)}.chosenval$ is the uniform distribution on D.

(b) If $u.Src_{cval}.chosenval = \bot$ then one of the following holds:

i. For every $s \in supp(lstate(\epsilon_1))$, for $i \in \{0,1\}$, $s.Src_{bvali} = \bot$.

ii. For some $i \in \{0,1\}$: for every $s \in supp(lstate(\epsilon_1))$, $s.Src_{bvali} = \bot$, and $s.Src_{bval(1-i)}.chosenval$ is the uniform distribution on $\{0,1\}$.

Lemma 9.10 *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, for every S, T, $|corrtasks(S,T)| \leq 1$.*

Proof. We show that R satisfies the two conditions in Lemma 3.55.
Start condition: It is obvious that the Dirac measures on execution fragments consisting of the unique start states s and u of $SInt2\|Env'$ and $SHROT\|Env'$, respectively, are R-related. The two properties of R hold because the state components of s and u on which R depends are all $\bot$.
Step condition: We define $corrtasks(RS_{SHROT\|Env'} \times RA_{SHROT\|Env'}) \to RA^*_{SInt2\|Env'}$ as follows:
For any $(S,T) \in (RS_{SHROT\|Env'} \times RA_{SHROT\|Env'})$:

- If $T = \{in(x)_{Trans}\}$ then $corrtasks(S,T) = T$.
- If $T = \{choose-rand_{tdp}\}$ then $corrtasks(S,T) = \{choose-rand_{tdpp}\}$.
- If $T = \{rand(f)_{tdp}\}$ then $corrtasks(S,T) = \{rand(p)_{tdpp}\}$.
- For $i \in \{0,1\}$:
  1. If $T = \{choose-rand_{zval(i)}\}$ and $s.Src_{zval(1-i)}.chosenval = \bot$ in every state s of S then $corrtasks(S,T) = \lambda$.
  2. If $T = \{choose-rand_{zval(i)}\}$ and $s.Src_{zval(1-i)}.chosenval \neq \bot$ in every state s of S then $corrtasks(S,T) = \{choose-rand_{zval}\}$.
  3. If $T = \{rand(z)_{zval(i)}\}$ and $s.Ifc.zval(1-i) = \bot$ in every state s of S then $corrtasks(S,T) = \lambda$.
  4. If $T = \{rand(z)_{zval(i)}\}$ and $s.Ifc.zval(1-i) \neq \bot$ in every state s of S then $corrtasks(S,T) = \{rand(z)_{zval}\}$.

- For $i \in \{0,1\}$:
  1. If $T = \{choose - rand_{bval(i)}\}$ and $s.Src_{bval(1-i)}.chosenval = \bot$ in every state s of S then $corrtasks(S,T) = \lambda$.
  2. If $T = \{choose - rand_{bval(i)}\}$ and $s.Src_{bval(1-i)}.chosenval \neq \bot$ in every state s of S then $corrtasks(S,T) = \{choose - rand_{cval}\}$.
  3. If $T = \{rand(b)_{bval(i)}\}$ and $s.Ifc.bval(1-i) = \bot$ in every state s of S then $corrtasks(S,T) = \lambda$.
  4. If $T = \{rand(b)_{bval(i)}\}$ and $s.Ifc.bval(1-i) \neq \bot$ in every state s of S then $corrtasks(S,T) = \{rand(c)_{cval}\}$.
- If $T = \{fix - bxorx\}$ then $corrtasks(S,T) = \{fix - bval_{Trans}\}$.
- If $T \in \{\{send(1,f)_{Trans}\}, \{send(2,z)_{Rec}\}, \{send(3,b)_{Trans}\}\}$ then $corrtasks(S,T) = T$.

The only interesting cases are those corresponding to the selection and to the transmission of $s.Src.zval(i).chosenval$ and $s.Src.bval(i).chosenval$. Each of these pairs of elements are selected into two random sources in $SHROT$ while they are selected as pairs of random elements into a single random source in $SInt2$.

We manage these differences in a simple way: when the first element of a pair is selected (resp. transmitted) in $SHROT$, we do not define any corresponding steps in $SInt2$, while the pairs are selected (resp. transmitted) in $SInt2$ when the second element of the pair is selected (resp. transmitted) in $SHROT$. This way to proceed will not raise any problem as all tasks of *Ifc* depending on these values are enabled only when both z-values and b-values have been transmitted.

Suppose $(\epsilon_1, \epsilon_2) \in R$ and T is a task of $SHROT \| Env'$ that is enabled in $supp(lstate(\epsilon_1))$. 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 9.7. State equivalence follows as in that proof. 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 (and S when needed).

1. $T = \{in(x)_{Trans}\}$ then $corrtasks(S,T) = \{in(x)_{Trans}\}$

   The treatment of this case is similar as the one described in the proof of Lemma 9.7.

2. $T = \{choose - rand_{tdp}\}$

   The treatment of this case is similar as Case 2 in the proof of Lemma 9.9.

3. $T = \{rand(f)_{tdp}\}$

   The treatment of this case is similar as Case 4 in the proof of Lemma 9.7.

4. $T = \{choose - rand_{zval(i)}\}$ and $s.Src_{zval(1-i)}.chosenval = \bot$ in every state s of S.

   There is no enabling condition to check since $corrtasks(S,T) = \lambda$.

   Let p be the Dirac measure on the single index 1 and let $\epsilon'_{11} = \epsilon'_1$, and $\epsilon'_{21} = \epsilon'_2 = \epsilon_2$. To show that $\epsilon'_1 R \epsilon'_2$, we establish Properties 1 and 2 of R for ϵ'_1 and ϵ'_2. Trace equivalence is immediate since T is an internal task.

   We observe that $\epsilon'_1 R \epsilon'_2$ because, when executing T the only modified state variable is $s.Src_{zval(i)}.chosenval$, Properties 1d, 1e are true because $s.Src_{zval(i)}.chosenval$ remains $\bot$, Property 2(a)i was true in all states of $supp(lstate(\epsilon_1))$, which implies that 2(a)ii is true in all states of $supp(lstate(\epsilon'_1))$.

5. $T = \{choose - rand_{zval(i)}\}$ and $s.Src_{zval(1-i)}.chosenval \neq \bot$ in every state s of S.

   We first need to check that $corrtasks(S,T) = \{choose - rand_{zval}\}$ is enabled in all states of $support(lstate(\epsilon_2))$. The fact that T is enabled in all states of $support(lstate(\epsilon_1))$ and Property 1d imply that $u.Src_{zval}.chosenval = \perp$ in all states of $support(lstate(\epsilon_2))$, which is the condition required for the $\{choose - rand_{zval}\}$ task to be enabled.

Let p be the uniform probability measure on the index set $I = \{1 \cdots r\}$ where $r = |D|^2$; that is, $p(j) = 1/r$ for each $j \in I$. For each $j \in I$, we define the probability measure ϵ'_{1j} as follows. The support $supp(\epsilon'_{1j})$ is the set of execution fragments $\alpha \in supp(\epsilon'_1)$ such that $(lstate(\alpha).Src_{zval0}.chosenval, lstate(\alpha).Src_{zval1}.chosenval)$ is the j-th element in domain $\{0,1\} \rightarrow D$ (in some fixed enumeration). For each $\alpha \in supp(\epsilon'_1)$ of the form $\alpha' choose{-}rand_{zval(i)} q$, let $\epsilon'_{1j} = \epsilon'_1(\alpha)$. This is acceptable since both $lstate(\alpha).Src_{zval0}.chosenval$ and $lstate(\alpha).Src_{zval1}.chosenval$ are the uniform distribution on D.

Now, we define ϵ'_{2j} from ϵ'_2. The support $supp(\epsilon'_{2j})$ is the set of execution fragments $\alpha \in supp(\epsilon'_2)$ such that $lstate(\alpha).Src_{zval}.chosenval$ is the j-th element in domain $\{0,1\} \rightarrow D$, keeping the same enumeration as in the previous paragraph. For each $\alpha \in supp(\epsilon'_2)$ of the form $\alpha' choose - rand_{zval} q$, let $\epsilon'_{2j} = \epsilon'_2(\alpha)$. Again, this is acceptable since $lstate(\alpha).Src_{zval}.chosenval$ is the uniform distribution on $\{0,1\} \rightarrow D$.

The trace equivalence property of R is preserved because both T and $corrtasks(S,T)$ are internal tasks. We now verify that $\epsilon'_{1j} R \epsilon'_{2j}$ for every j. Fix some $j \in I$. The only changing variables in $supp(lstate(\epsilon_1))$ and $supp(lstate(\epsilon_2))$ are $s.Src_{zval(i)}.chosenval$ and $u.Src_{zval}.chosenval$. Property 1d remains true because $s.Src_{zval0}.chosenval$, $s.Src_{zval1}.chosenval$ and $u.Src_{zval}.chosenval$ are all different from $\perp$ now. Property 1e remains true because we are using one single enumeration of the elements of $\{0,1\} \rightarrow D$. Eventually, Property 2a holds because $u.Src_{zval}.chosenval \neq \perp$ in all states of $supp(lstate(\epsilon_2))$.

6. $T = \{rand(z)_{zval(i)}\}$ and $s.Ifc.zval(1-i) = \perp$ in every state s of S.

   Applying T and $corrtasks(S,T)$ may have two types of consequences. First, it might be the case that T has already been applied before. In that case, $supp(lstate(\epsilon_1)) = supp(lstate(\epsilon'_1))$, $supp(lstate(\epsilon_2)) = supp(lstate(\epsilon'_2))$, and it is immediate that the different conditions on R still hold.

   On the other hand, if $s.Ifc.zval(i) = \perp$ in every state s of S, applying T and $corrtasks(S,T)$ only changes $s.Ifc.zval(i)$ in all states s of $supp(lstate(\epsilon_1))$. We observe that $s.Ifc.zval(i)$ only appears in Properties 1h and 1i, and these properties remain true after applying those tasks because we know that $s.Ifc.zval(1-i) = \perp$ in every state s of S.

7. $T = \{rand(z)_{zval(i)}\}$ and $s.Ifc.zval(1-i) \neq \perp$ in every state s of S.

   Again, applying T and $corrtasks(S,T)$ may have two types of consequences. First, it might be the case that T has already been applied before and, in this case, the considerations of the previous paragraph still apply.

   On the other hand, if $s.Ifc.zval(i) = \perp$ in every state s of S, applying T and $corrtasks(S,T)$ only changes $s.Ifc.zval(i)$ in all states s of $supp(lstate(\epsilon_1))$ and $u.TR2.zval$. We observe that these variables only appears in Properties 1h and 1i, and these properties remain true after applying those tasks because $u.TR2.zval = s.Ifc.zval$ in all states $s \in supp(lstate(\epsilon'_1))$ and $u \in supp(lstate(\epsilon'_2))$, as guaranteed by Property 1e of relation R and Lemma 8.13.

8. $T = \{choose - rand_{bval(i)}\}$, $T = \{rand(b)_{bval(i)}\}$.

   The treatment of the four corresponding cases is similar to those corresponding to the tasks $\{choose - rand_{zval(i)}\}$ and $\{rand(z)_{zval(i)}\}$.

9. $T = \{fix - bxorx\}$.

   We first need to check that $corrtasks(S,T) = \{fix - bval_{Trans}\}$ is enabled in all states of $support(lstate(\epsilon_2))$. Since T is enabled in all states of $support(lstate(\epsilon_1))$, we know that $s.Ifc.bval \neq \perp$, $s.Ifc.xval \neq \perp$, $s.Ifc.fval \neq \perp$, $s.Ifc.zval \neq \perp$ and $s.Ifc.bxorx = \perp$. Now, Properties 1j and 1k of R imply that $u.TR2.cval \neq \perp$, Property 1a of R implies that

$u.TR2.inval(Trans) \neq \perp$, Property 1c of R implies that $u.TR2.tdpp \neq \perp$, Property 1h of R implies that $u.TR2.zval \neq \perp$ and Property 1l of R implies that $u.TR2.bval = \perp$. We can check that these properties guarantee that $\{fix - bval_{Trans}\}$ is enabled in all states of $support(lstate(\epsilon_2))$.

Finally, the definition of the $fix - bxorx$ and $fix - bval_{Trans}$ actions show that the only state variables modified by the corresponding tasks are $s.Ifc.bxorx$ and $u.Ifc.bval$, which only appear in Property 1l of R. It is easy to see that this property remains verified by using Properties 1a and 1k of R and observing that $s.Ifc.bxorx$ and $u.Ifc.bval$ are computed in the same way from equal state variables.

10. $T \in \{\{send(1, f)_{Trans}\}, \{send(2, z)_{Rec}\}, \{send(3, b)_{Trans}\}\}$.

    The treatment of these cases is similar to the corresponding ones in the proof of Lemma 9.7.

□

9.5.3 *Int1* implements *Int2*

Proof. (of Lemma 9.8)
In Lemma 9.9 and 9.10, 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. 3.86 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 8.11) and since the $\leq_{neg,pt}$ implementation relation is transitive (see Lemma 3.83), 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 3.84, 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:

$$\begin{aligned}&hide_{\{rand(*)_{tdpp}\} \cup \{rand(*)_{zval}\}}(\overline{Funct} \| \overline{Adv} \| \overline{TR1} \| \overline{Src_{tdpp}} \| \overline{Src_{zval}})\\ &\quad \leq_{neg,pt} hide_{\{rand(*)_{tdpp}\} \cup \{rand(*)_{zval}\} \cup \{rand(*)_{cval}\}}(\overline{Funct} \| \overline{Adv} \| \overline{TR2} \| \overline{Src_{tdpp}} \| \overline{Src_{zval}} \| \overline{Src_{cval}}),\end{aligned}$$

or in other words, $\overline{Int1} \leq_{neg,pt} \overline{Int2}$, as needed. □

9.6 *Int2* implements SIS

We show:

Lemma 9.11 *For every k, $Int2_k \leq_0 SIS_k$.*

We prove Lemma 9.11 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 3.53, the soundness result for simulation relations.

The only differences between *Int2* and *SIS* are that *Int2* uses *TR2* and Src_{cval} whereas *SIS* uses *TR* and Src_{bval}. The key difference here is that *TR2* calculates the *bval* values as $\oplus$'s of random *cval* values and the input x values, whereas *TR* just chooses the *bval* values randomly. However, since taking $\oplus$ with a random bit is the same as choosing a random bit, this does not give any observably-different behavior.

We also show:

Lemma 9.12 $\overline{Int2} \leq_{neg,pt} \overline{SIS}$.

In the rest of this subsection, we fix *Env*, an environment for $Int2_k$ and SIS_k. We also suppress mention of k everywhere.

9.6.1 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 9.6.2.

Let ϵ_1 and ϵ_2 be discrete probability measures on finite execution fragments of $Int2 \| Env$ and $SIS \| Env$, respectively, satisfying the following properties:

1. **Trace distribution equivalence:** $tdist(\epsilon_1) = tdist(\epsilon_2)$.
2. **State equivalence:** There exist state equivalence classes $S_1 \in RS_{Int2 \| Env}$ and $S_2 \in RS_{SIS \| Env}$ such that $supp(lstate(\epsilon_1)) \subseteq S_1$ and $supp(lstate(\epsilon_2)) \subseteq S_2$.

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) $u.Funct.inval(Trans) = s.TR2.inval(Trans)$.
   (c) $u.TR.tdpp = s.TR2.tdpp$.
   (d) $u.TR.zval = s.TR2.zval$.
   (e) $u.TR.bval = s.TR2.bval$.
   (f) $u.Src_{tdpp} = s.Src_{tdpp}$.
   (g) $u.Src_{zval} = s.Src_{zval}$
   (h) $u.Src_{bval}.chosenval = s.TR2.bval$.
   (i) $u.Adv = s.Adv$.
   (j) $u.Env = s.Env$.
2. For every $u \in supp(lstate(\epsilon_2))$:
   If $u.TR.bval = \perp$ then one of the following holds:
   (a) For every $s \in supp(\epsilon_1)$, $s.Src_{cval}.chosenval = \perp$.
   That is, *cval* has not yet been chosen.
   (b) For every $s \in supp(\epsilon_1)$, $s.TR2.cval = \perp$, and $lstate(\epsilon_1).Src_{cval}.chosenval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$.
   That is, *cval* has been chosen by the *Src*, but has not yet been output to *TR2*.
   (c) $lstate(\epsilon_1).TR2.cval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$.

9.6.2 The mapping proof

Lemma 9.13 *The relation R defined in Section 9.6.1 is a simulation relation from $Int2 \| Env$ to $SIS \| Env$. Furthermore, for each step of $Int2 \| Env$, the step correspondence yields at most two steps of $SIS \| Env$, that is, for every S, T, $|corrtasks(S,T)| \leq 2$.*

Proof. We prove that R satisfies the two conditions in Lemma 3.55.
Start condition: It is obvious that the Dirac measures on execution fragments 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 $s.Src_{cval}.chosenval = \perp$.

Step condition: We define $corrtasks : RS_{Int2 \| Env} \times RA_{Int2 \| Env} \rightarrow RA^*_{SIS \| Env}$ as follows:
For any $(S,T) \in RS_{Int2 \| Env} \times RA_{Int2 \| Env}$:

- If $T \in \{\{in(x)_{Trans}\}, \{in(i)_{Rec}\}, \{choose-rand_{tdpp}\}, \{rand_{tdpp}\}, \{choose-rand_{zval}\}, \{rand_{zval}\}$ $\{send(1,f)_{Trans}\}, \{receive(1,f)_{Rec}\}, \{send(2,z)_{Rec}\}, \{receive(2,z)_{Trans}\}, \{send(3,b)_{Trans}\},$ $\{receive(3,b)_{Rec}\}$, or $\{out(x)_{Rec}\}\}$, then $corrtasks(S,T) = T$.

- If T is an output or internal task of *Env* or *Adv* that is not one of the tasks listed above, then $corrtasks(S,T) = T$.
- If $T \in \{\{choose-rand_{cval}\}, \{rand_{cval}\}\}$ then $corrtasks(S,T) = \lambda$.
- If $T = \{fix-bval_{Trans}\}$ then $corrtasks(S,T) = \{choose-rand_{bval}\}$ $\{rand_{bval}\}$.

Suppose $(\epsilon_1, \epsilon_2) \in R$ and T is a task of RS_{Int2} that is enabled in $supp(lstate(\epsilon_1))$. 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 9.7. State equivalence follows as in that proof. Identical versions of Claim 1 and Claim 2 in that proof carry over to this case. We again consider cases based on the value of T.

1. $T = \{in(x)_{Trans}\}$.

   Since T is an output task of *Env*, 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 $tr_{Env} = (q_{Env}, a, \mu_{Env})$ of *Env* from q_{Env} with action a. Here, $a = in(x)_{Trans}$ for a particular value of x.

   Next 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 execution fragments $\alpha \in supp(\epsilon'_1)$ 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 $j \in I$; we show that $(\epsilon'_{1j}, \epsilon'_{2j}) \in R$. To do this, we establish Properties 1 and 2 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 $(u, a, \mu_u) \in D_{SIS\|Env}$.

   If $s.TR2.inval(Trans) \neq \bot$ then by Properties 1(a) and 1(b), $u.Funct.inval(Trans) \neq \bot$ and $s.Funct.inval(Trans) \neq \bot$. 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'.

   Now suppose that $s.TR2.inval(Trans) = \bot$. Then again by Properties 1(a) and 1(b), $u.Funct.inval(Trans) = s.Funct.inval(Trans) = \bot$. Then by the definitions of *Int2* and *SIS*, we know that application of T updates $TR2.inval(Trans)$ and $Funct.inval(Trans)$ in *Int2*, and $Funct.inval(Trans)$ in *SIS*. It also updates the state of *Env* to q_j in both systems.

   We know by Property 1(a) that $u.Funct = s.Funct$, by Property 1(b) that $u.Funct.inval(Trans) = s.TR2.inval(Trans)$, and by 1(j) that $u.Env = s.Env$. By the effects of T in definitions of *Funct* and *TR2*, we know that $u'.Funct = s'.Funct$ and $u'.Funct.inval(Trans) = s'.TR2.inval(Trans)$; hence, Properties 1(a) and 1(b) hold for s' and u'. We also know that 1(j) holds for s' and u' by definition of ϵ'_{1j} and ϵ'_{2j}: in both s' and u', the state of *Env* is q_j. Since no state component other than $TR2.inval$, $Funct.inval(Trans)$, and *Env* in the *TR2* system, and $Funct.inval(Trans)$ and *Env* 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, consider any state $u' \in supp(lstate(\epsilon'_{2j}))$ such that $u'.TR.bval = \bot$. We need to show that one of the following holds:

   (a) For every $s' \in supp(lstate(\epsilon'_{1j}))$, $s'.Src_{cval}.chosenval = \bot$.

   (b) For every $s' \in supp(lstate(\epsilon'_{1j}))$, $s'.TR2.cval = \bot$, and $lstate(\epsilon'_{1j}).Src_{cval}.chosenval$ is the uniform distribution on $\{0,1\} \to \{0,1\}$.

(c) $lstate(\epsilon'_{1j}).TR2.cval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$.

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 T, we know that $u.TR.bval = u'.TR.bval = \bot$. Then, by Property 2 for ϵ_1 and u, one of the following holds:

(a) For every $s \in supp(lstate(\epsilon_1))$, $s.Src_{cval}.chosenval = \bot$.

(b) For every $s \in supp(lstate(\epsilon_1))$, $s.TR2.cval = \bot$, and $lstate(\epsilon_1).Src_{cval}.chosenval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$.

(c) $lstate(\epsilon_1).TR2.cval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$.

If (a) holds for ϵ_1 and u, then consider any $s' \in supp(lstate(\epsilon'_{1j}))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, a, \mu_s) \in D_{Int2\|Env}$. We have by (a) that $s.Src_{cval}.chosenval = \bot$. By the effects of T, $s'.Src_{cval}.chosenval = s.Src_{cval}.chosenval = \bot$, and so (a) holds for ϵ'_{1j} and u'.

If (b) holds for ϵ_1 and u, then consider any $s' \in supp(lstate(\epsilon'_{1j}))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, a, \mu_s) \in D_{Int2\|Env}$. By the effects of T, $s'.TR2.cval = s.TR.cval = \bot$, so the first part of (b) holds. For the second part of (b), 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 $\alpha'\, a\, q$, we have $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$. Since T transitions do not affect the value of $Src_{cval}.chosenval$, we have that $lstate(\epsilon'_{1j}).Src_{cval}.chosenval = lstate(\epsilon_1).Src_{yval}.chosenval$, and (b) holds for ϵ'_{1j} and u'.

If (c) holds for ϵ_1 and u, then we argue as for the second part of (b), using the fact that T transitions do not affect $TR2.cval$. Thus, (c) holds for ϵ'_{1j} and u'. Therefore, in all cases, Property 2 holds for ϵ'_{1j} and u', and hence for ϵ'_{1j} and ϵ'_{2j}.

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. $T = \{in(i)_{Rec}\}$.

   Since T is an output task of Env, 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 $tr_{Env} = (q_{Env}, a, \mu_{Env})$ of Env from q_{Env} with action a. Here, $a = in(i)_{Rec}$ for a particular value of i.

   The rest of the proof for this case follows the proof for $T = \{in(x)_{Trans}\}$. The only difference is that, in showing that Property 1 holds for ϵ'_{1j} and ϵ'_{2j}, for a fixed j, we use the fact that application of T affects only $Funct.inval(Rec)$ and Env in the $Int2$ system and the SIS system, and use Properties 1(a) and 1(j).

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

   Identical to the corresponding case in the proof of Lemma 9.7, but using Property 1(f) instead of 1(g).

4. $T = \{rand(p)_{tdpp}\}$.

   Identical to the corresponding case in the proof of Lemma 9.7, but using Properties 1(c) and 1(f) instead of 1(d) and 1(g).

5. $T = \{choose-rand_{zval}\}$.

   Identical to the proof for $T = \{choose-rand_{tdpp}\}$, but using Property 1(g) instead of 1(f).

6. $T = \{rand(z)_{zval}\}$.

   Identical to the proof for $T = \{rand(z)_{tdpp}\}$, but using Properties 1(d) and 1(g) instead of 1(c) and 1(f).

7. $T = \{choose - rand_{cval}\}$.

   Here, a random choice is made in the *Int2* system but not in the *SIS* system. Since $corrtasks([lstate(\epsilon_1)], T) = \lambda$, no enabling condition needs to be shown. Also, we have $\epsilon'_2 = \epsilon_2$.

   Next, we define the probability measures. Let p be the Dirac measure on the single index 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 $u' \in supp(lstate(\epsilon'_2))$. Since $\epsilon'_2 = \epsilon_2$, we know that $u' \in supp(lstate(\epsilon_2))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$, where $(s, choose - rand_{cval}, \mu_s) \in D_{Int2\|Env}$. We know that Property 1 holds for s and u'. Observe that the application of T updates only $s.Src_{cval}.chosenval$ component in the RS system, and the application of λ leaves u' unchanged. Since Property 1 does not mention $Src_{cval}.chosenval$, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.

   To establish Property 2, consider any state $u' \in supp(lstate(\epsilon'_2))$ such that $u'.TR.bval = \perp$. We show that Property 2(b) holds; that is, we show that for every $s' \in supp(lstate(\epsilon'_1))$, $s'.TR2.cval = \perp$, and $lstate(\epsilon'_1).Src_{cval}.chosenval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$.

   Consider any $s' \in supp(lstate(\epsilon'_1))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, choose - rand_{cval}, \mu_s) \in D_{Int2\|Env}$. Since $choose - rand_{cval}$ is enabled in s, we know that $s.Src_{cval}.chosenval = \perp$. Therefore, by Lemma 9.4, $s.TR2.cval = \perp$. Since T does not update $TR2.cval$, we have $s'.TR2.cval = \perp$. Hence, the first part of 2(b) holds.

   For the second part of 2(b), the effects of T imply that $Src_{cval}.chosenval$ is chosen according to the uniform probability distribution on domain $\{0,1\} \rightarrow D$. So, $lstate(\epsilon'_1).Src_{cval}.chosenval$ gives the uniform distribution on $\{0,1\} \rightarrow D$, as needed.

   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. $T = \{rand(c)_{cval}\}$.

   This is a case where a step is taken in the *Int2* system but not in the *SIS* system. Since $corrtasks([lstate(\epsilon_1)], T) = \lambda$, no enabling condition needs to be shown, and $\epsilon'_2 = \epsilon_2$.

   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 $\epsilon'_{21} = \epsilon'_2$.

   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.

   To establish Property 1, consider any states $s' \in supp(lstate(\epsilon'_1))$ and $u' \in supp(lstate(\epsilon'_2))$. Since $\epsilon'_2 = \epsilon_2$, we know that $u' \in supp(lstate(\epsilon_2))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, rand(c)_{cval}, \mu_s) \in D_{Int2\|Env}$ and $c = (s.Src_{cval}.chosenval)$. We know that Property 1 holds for s and u'. Observe that the application of T updates only the $s.TR2.cval$ component in the *Int2* system and the application of λ leaves u' unchanged. Since Property 1 does not mention $TR2.cval$, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2, as needed.

   To establish Property 2, consider any state $u' \in supp(lstate(\epsilon'_2))$ such that $u'.TR.bval = \perp$. We show that Property 2(c) holds; that is, we show that $lstate(\epsilon'_1).TR2.cval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$.

   Since $u' \in supp(lstate(\epsilon_2))$, we know that Property 2 holds for u' and ϵ_1. However, 2(a) cannot hold because T is enabled in $supp(lstate(\epsilon_1))$, so either 2(b) or 2(c) must hold for u' and ϵ_1.

   If 2(b) holds for u' and ϵ_1, then consider any $s' \in supp(lstate(\epsilon'_1))$. Let s be any state in $supp(lstate(\epsilon_1))$ such that $s' \in supp(\mu_s)$ where $(s, rand(c)_{cval}, \mu_s) \in D_{Int2\|Env}$ and $c = s.Src_{cval}.chosenval$. We know that $s.TR2.cval = \perp$ and $lstate(\epsilon_1).Src_{cval}.chosenval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$. Then, by the effects of T and the definition of ϵ'_1,

$s'.TR2.cval \neq \perp$ and $lstate(\epsilon_1').TR2.cval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$, and hence 2(c) holds for u' and ϵ_1', as needed.

On the other hand, if Property (c) holds for u' and ϵ_1, then we know that $lstate(\epsilon_1)$ projected on $TR2.cval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$. Since the application of T affects $TR2.cval$ only if it is $\perp$, we know that $lstate(\epsilon_1').TR2.cval = lstate(\epsilon_1).TR2.cval$. Therefore, in this case 2(c) holds for u' and ϵ_1', as needed.

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'.

9. $T = \{fix - bval_{Trans}\}$.

   Here, a deterministic step in the *Int2* system maps to a random choice followed by a deterministic step in the SIS system. We first show that the sequence of tasks $\{choose-rand_{bval}\}\{rand(b)_{bval}\}$ is enabled in $supp(lstate(\epsilon_2))$. First, consider any state $u \in supp(lstate(\epsilon_2))$; we show that $\{choose - rand_{bval}\}$ 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 $TR2$, T is enabled in $s.TR2$. By the precondition of the $fix - bval_{Trans}$ action in $TR2$, we know that $s.TR2.bval = \perp$. By Property 1(h) for s and u, $u.Src_{bval}.chosenval = \perp$. So, $\{choose - rand_{zval}\}$ is enabled in u, as needed.

   Now, let ϵ_2'' be the measure $apply(\epsilon_2, \{choose - rand_{bval}\})$. We show that $\{rand(b)_{bval}\}$ is enabled in $supp(lstate(\epsilon_2''))$. So consider any state $u'' \in supp(lstate(\epsilon_2''))$. By the effect of $\{choose - rand_{bval}\}$, we know that $u''.Src_{bval}.chosenval \neq \perp$, which is the only precondition on actions in $\{rand(b)_{bval}\}$. Thus, $\{rand(b)_{bval}\}$ is enabled in $supp(lstate(\epsilon_2''))$, as needed.

   Next, we claim that $lstate(\epsilon_1).TR2.cval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$. To see this, consider any pair of states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$. Since $s.TR2.bval = \perp$, by Property 1(e) we have $u.TR.bval = \perp$. Then by Property 2 for u and ϵ_1, we know that one of the following holds:

   (a) $s.Src_{cval}.chosenval = \perp$.

   (b) $s.TR2.cval = \perp$ and $lstate(\epsilon_1).Src_{cval}.chosenval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$.

   (c) $lstate(\epsilon_1).TR2.cval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$.

   However, since T is enabled in $supp(lstate(\epsilon_1))$, we know that $s.TR2.cval \neq \perp$, so 2(b) cannot hold. Using Lemma 9.4, we see that also 2(a) cannot hold. Therefore, 2(c) holds, that is, $lstate(\epsilon_1).TR2.cval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$, as needed.

   Next, we show that $lstate(\epsilon_1').TR2.bval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$. By Property 1(b), $inval(Trans)$ is the same in all states in $supp(lstate(\epsilon_1))$. The effect of a $fix - bval_{Trans}$ action in $TR2$ is to assign $TR2.bval$ a pair of bits obtained by applying $\oplus$ to The $cval$ bits and the $inval(Trans)$ bits. Thus, since $lstate(\epsilon_1).TR2.cval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$, it follows that $lstate(\epsilon_1').TR2.bval$ is the uniform distribution on $\{0,1\} \rightarrow \{0,1\}$.

   Next 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 $r = |\{0,1\} \rightarrow \{0,1\}| = 4$. That is, $p(j) = 1/4$ 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 execution fragments $\alpha \in supp(\epsilon_1')$ such that $lstate(\alpha).TR2.bval$ is the jth element of the domain $\{0,1\} \rightarrow \{0,1\}$. For each $\alpha \in supp(\epsilon_{1j}')$ of the form $\alpha' \, fix - bval_{Trans} \, q$, let $\epsilon_{1j}'(\alpha) = \epsilon_1(\alpha')$. Similarly, we define probability distribution ϵ_{2j}' as follows. The support $supp(\epsilon_{2j}')$ is the set of execution fragments $\alpha \in supp(\epsilon_2')$ such that $lstate(\alpha).TR2.bval$ is the jth element of the domain $\{0,1\} \rightarrow \{0,1\}$. For each $\alpha \in supp(\epsilon_{2j}')$ of the form $\alpha' \, choose - rand_{bval} \, q \, rand(b)_{bval} q'$ let $\epsilon_{2j}'(\alpha) = \epsilon_2(\alpha')$.

   Now fix $j \in I$; we show that $(\epsilon_{1j}', \epsilon_{2j}') \in R$. To do this, we establish Properties 1 and 2 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 *Int2* and *SIS* we know that application of T updates $TR2.bval$ in the *Int2* system and application of the sequence $\{choose-rand_{bval}\}\,\{rand(b)_{bval}\}$ updates $Src_{bval}.chosenval$ and $TR.bval$ in the *SIS* system. We show that Properties 1(e) and 1(h) hold for u' and s'.

Property 1(e) follows from the definitions of ϵ'_{1j} and ϵ'_{2j}; both actions give the same element of the domain $\{0,1\} \rightarrow \{0,1\}$ when projected onto $TR2.bval$ and $TR.bval$. For Property 1(h), we use the fact that $u'.TR.bval = s'.TR2.bval$, and we observe in addition (using Lemma 9.2) that if $u'.TR.bval \neq \perp$ then $u'.Src_{bval}.chosenval = u'.TR.bval$. Since no state component other than $TR2.bval$ in the *Int2* system is updated by the application of T, and no state component other than $TR.bval$ and $Src_{bval}.chosenval$ is updated by the application of $\{choose-rand_{bval}\}\,\{rand(b)_{bval}\}$ in the *SIS* system, we conclude that Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.

Property 2 holds trivially in this case since for any state $u' \in supp(lstate(\epsilon'_{2j}))$, we have $u'.TR.bval \neq \perp$ by definition of ϵ'_{2j}.

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}.

10. $T = \{send(1, f)_{Trans}\}$.

    Identical to the corresponding case in the proof of Lemma 9.7, except that here we replace *Trans* with *TR2* and *TR1* with *TR*, and use Properties 1(c) and 1(i) instead of 1(d) and 1(i).

11. $T = \{send(2, z)_{Rec}\}$.

    Identical to the corresponding case in the proof of Lemma 9.7, except that here we replace *Rec* with *TR2* and *TR1* with *TR*, and use Property 1(d) instead of 1(e).

12. $T = \{send(3, b)_{Trans}\}$.

    Identical to the corresponding case in the proof of Lemma 9.7, except that here we replace *Trans* with *TR2* and *TR1* with *TR*, and use Property 1(e) here instead of 1(f).

13. $T = \{receive(1, f)_{Rec}\}$.

    Identical to the corresponding case in the proof of Lemma 9.7, except that here we replace *Rec* with *TR2*. In showing Property 1, we use the fact that applying T has no effect in either system.

14. $T = \{receive(2, z)_{Trans}\}$.

    Identical to the corresponding case in the proof of Lemma 9.7, except that here we replace *Trans* with *TR2*. In showing Property 1, we use the fact that applying T has no effect in either system.

15. $T = \{receive(3, b)_{Rec}\}$.

    Identical to the corresponding case in the proof of Lemma 9.7, except that here we replace *Rec* with *TR2*. In showing Property 1, we use the fact that applying T has no effect in either system.

16. $T = \{out(x)_{Rec}\}$.

    This case is easier that its counterpart in the proof of Lemma 9.7, since the task is an output task from *Funct* to *Env* in both levels. We use Property 1(a) to show enabling. The only interesting aspect of this proof is that *Env* may make a probabilistic choice on the application of T. The step correspondence can be shown by decomposing the distributions generated by application of T as in the case for $T = \{in(x)_{Trans}\}$.

    We first show that T is enabled in every state in $supp(lstate(\epsilon_2))$. So, fix any state $u \in supp(lstate(\epsilon_2))$; we show that T is enabled in u. Note that T is an output task of *Funct* in both systems. Choose any $s \in supp(lstate(\epsilon_1))$. By Property 1(a), $u.Funct = s.Funct$. So, T is enabled in $u.Funct$, and hence in u, as needed.

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 1(a) that the state of *Funct* is the same in all states in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. So, $out(s.Funct.inval(Trans)(s.Funct.inval(Rec)))$ is the unique action in T that is enabled in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$. We use a to refer to $out(s.Funct.inval(Trans)(s.Funct.inval(Rec))$ in the rest of the proof for this case. Then next-transition determinism for *Env* implies that there is a unique transition of *Env* from q_{Env} with action a. Let $tr_{Env} = (q_{Env}, a, \mu_{Env})$ be this unique transition.

We define the probability measures needed to show the step correspondence as in the case for $in(x)_{Trans}$.

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 the definitions of the *Int2* and *SIS* systems, we know that application of T does not update any state component of *Int2* or *SIS*; however, it may update the state of *Env* in both systems. Since Property 1 holds for s and u, we know that all the parts of Property 1 except possible for 1(j) also hold for s' and u'. We also know that 1(j) holds for s' and u' by definition of ϵ'_{1j} and ϵ'_{2j}: in both s' and u', the state of *Env* is q_j. Thus, Property 1 holds for s' and u', and hence, for ϵ'_1 and ϵ'_2.

The proofs for Property 2 and trace distribution equivalence are similar to the corresponding parts of the proof for $T = \{in(x)_{Trans}\}$.

17. T is an output task of *Env* and an input task of *Adv*.

    Identical to the corresponding case in the proof of Lemma 9.7.

18. T is an output task of *Env* that is not an input task of *Adv*, *Funct*, or *TR2*, or T is an internal task of *Env*.

    Identical to the corresponding case in the proof of Lemma 9.7.

19. T is an output task of *Adv* and an input task of *Env*.

    Identical to the corresponding case in the proof of Lemma 9.7.

20. T is an output task of *Adv* that is not an input task of *Env*, *Funct*, or *TR2*, and is not a *receive* task, or else T is an internal task of *Adv*.

    Identical to the corresponding case in the proof of Lemma 9.7.

□

Proof. (Of Lemma 9.11:)
By Lemma 9.13, R is a simulation relation from $Int2_k\|Env$ to $SIS_k\|Env$. Then Theorem 3.53 implies that $tdists(Int2_k\|Env) \subseteq tdists(SIS_k\|Env)$. Since *Env* was chosen arbitrarily, this implies (by definition of $\leq_0$) that $RS_k \leq_0 Int1_k$. □

Proof. (Of Lemma 9.12:)
By Lemma 9.13, R is a simulation relation from $RS_k\|Env$ to $Int1_k\|Env$ for which $|corrtasks(S,T)| \leq 2$ for every S and T. Since that lemma holds for every k and every *Env*, Theorem 3.86 implies that $\overline{Int2} \leq_{neg,pt} \overline{SIS}$. □

9.7 Putting the pieces together

Proof. (of Theorem 9.1):
Lemmas 9.6, 9.8, and 9.12, and transitivity of $\leq_{neg,pt}$, imply that $\overline{RS} \leq_{neg,pt} \overline{SIS}$. Since the simulator $SSim_k$ satisfies the constraints for a simulator in Figure 2, this implies that $\overline{RS} \leq_{neg,pt} \overline{IS}$. □

10 Correctness Proof, Case 2: Receiver Corrupted

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 $\overline{RS} \leq_{neg,pt} \overline{IS}$.

As before, since $C = \{Rec\}$ everywhere in this section, we drop explicit mention of C. Again, we express each Sim_k as a composition of automata, and show that $\overline{RS}$, the real-system family, implements the (new) structured-ideal-system family $\overline{SIS}$. Again, we introduce two intermediate levels, $\overline{Int1}$ and $\overline{Int2}$, for the same purpose as in Section 9.

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'' actions 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 $Src(\{0,1\})$.
- 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 $out''(x)_{Rec}$.

TR has *send* outputs that are inputs to Adv'. The *receive* outputs of Adv' are not connected to anything.

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, and $out(x)_{Rec}$ outputs, which go to the environment. 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 16. 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 $TR_k.inval(Trans) = Funct_k.inval(Trans)(Funct_k.inval(Rec))$.*

$TR(D, Tdp)$:

Signature:

Input:
- $out'(x)_{Rec}$, $x \in \{0,1\}$
- $in(i)_{Rec}$, $i \in \{0,1\}$
- $rand(p)_{tdpp}$, $p \in Tdpp$
- $rand(y)_{yval}$, $y \in (\{0,1\} \rightarrow D)$
- $rand(b)_{bval1}$, $b \in \{0,1\}$

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

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

State:
- $inval(Trans), inval(Rec) \in \{0,1,\perp\}$, initially $\perp$
- $tdpp \in Tdp \cup \{\perp\}$, initially $\perp$
- $yval, zval \in (\{0,1\} \rightarrow D) \cup \{\perp\}$, initially $\perp$
- $bval1 \in \{0,1,\perp\}$, initially $\perp$
- $bval \in (\{0,1\} \rightarrow \{0,1\}) \cup \{\perp\}$, initially $\perp$

Transitions:

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

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

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

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

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

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

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

$out''(x)_{Rec}$
Precondition:
  $x = inval(Trans) \neq \perp$
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: $\{out'(*)_{Rec}\}$, $\{in(*)_{Rec}\}$, $\{rand(*)_{tdpp}\}$, $\{rand(*)_{yval}\}$, $\{rand(*)_{bval1}\}$, $\{send(1,*)_{Trans}\}$, $\{send(2,*)_{Rec}\}$, $\{send(3,*)_{Trans}\}$, $\{out''(*)_{Rec}\}$, $\{fix-zval_{Rec}\}$, $\{fix-bval_{Trans}\}$.

State relation: q_1 and q_2 are related iff:
$q_1.inval(Trans) = \perp$ iff $q_2.inval(Trans) = \perp$, and similarly for $inval(Rec)$, $tdpp$, $yval$, $zval$, $bval1$, and $bval$.

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

10.2 *Int1*

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 17. *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.

$TR1(D, Tdp)$:

Signature:

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

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 $\perp$
- $inval2(Trans) \in (\{0,1\} \to \{0,1\}) \cup \{\perp\}$, initially $\perp$
- $tdpp \in Tdpp \cup \{\perp\}$, initially $\perp$
- $yval, zval \in (\{0,1\} \to D) \cup \{\perp\}$, initially $\perp$
- $bval \in (\{0,1\} \to \{0,1\}) \cup \{\perp\}$, initially $\perp$

Transitions:

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

$out'(x)_{Rec}$, $in(i)_{Rec}$, $rand(p)_{tdpp}$, or $rand(y)_{yval}$
Effect:
  As for $TR(D, Tdp)$.

$fix - bval_{Trans}$
Precondition:
  $tdpp, zval, inval(Trans), inval2(Trans), inval(Rec) \neq \perp$
  $bval = \perp$
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: $\{in(*)_{Trans}\}$, $\{out'(*)_{Rec}\}$, $\{in(*)_{Rec}\}$, $\{rand(*)_{tdpp}\}$, $\{rand(*)_{yval}\}$, $\{send(1,*)_{Trans}\}$, $\{send(2,*)_{Rec}\}$, $\{send(3,*)_{Trans}\}$, $\{out''(*)_{Rec}\}$, $\{fix - zval_{Rec}\}$, $\{fix - bval_{Trans}\}$.

State relation: q_1 and q_2 are related iff:
$q_1.inval(Trans) = \perp$ iff $q_2.inval(Trans) = \perp$, and similarly for $inval(Rec)$, $inval2(Trans)$, $tdpp$, $yval$, $zval$, and $bval$.

Figure 17: $TR1(D, Tdp)$, for the case where $C = \{Rec\}$.

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 $TR1_k.inval(Rec) \neq \perp$.*

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 $cval1 := c$.
   (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 18.

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

We show:

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 3.53, the soundness result for simulation relations. As for Case 1, 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 execution fragments of $RS \| Env$ and $Int1 \| Env$, respectively satisfying the following properties:

1. **Trace distribution equivalence:** $tdist(\epsilon_1) = tdist(\epsilon_2)$.
2. **State equivalence:** There exist state equivalence classes $S_1 \in RS_{RS\|Env}$ and $S_2 \in RS_{Int1\|Env}$ such that $supp(lstate(\epsilon_1)) \subseteq S_1$ and $supp(lstate(\epsilon_2)) \subseteq S_2$.

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

$TR2(D, Tdp)$:

Signature:

Input:

$in(x)_{Trans}$, $x \in (\{0,1\} \rightarrow \{0,1\})$
$out'(x)_{Rec}$, $x \in \{0,1\}$
$in(i)_{Rec}$, $i \in \{0,1\}$
$rand(p)_{tdpp}, p \in Tdpp$
$rand(y)_{yval}, y \in (\{0,1\} \rightarrow D)$
$rand(c)_{cval1}, c \in \{0,1\}$

Output:

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

Internal:

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

State:

$inval(Trans), inval(Rec) \in \{0,1,\perp\}$, initially $\perp$
$inval2(Trans) \in (\{0,1\} \rightarrow \{0,1\}) \cup \{\perp\}$, initially $\perp$
$tdpp \in Tdpp \cup \{\perp\}$, initially $\perp$
$yval, zval \in (\{0,1\} \rightarrow D) \cup \{\perp\}$, initially $\perp$
$bval \in (\{0,1\} \rightarrow \{0,1\}) \cup \{\perp\}$, initially $\perp$
$cval1 \in \{0,1,\perp\}$, initially $\perp$.

Transitions:

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

$out'(x)_{Rec}$, $in(i)_{Rec}$, $rand(p)_{tdpp}$, or $rand(y)_{yval}$
Effect:
As for $TR(D, Tdp)$.

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

$fix-bval_{Trans}$
Precondition:
$yval, cval1, inval(Trans), inval2(Trans) \neq \perp$
$inval(Rec) \neq \perp$
$bval = \perp$
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: $\{in(*)_{Trans}\}$, $\{out'(*)_{Rec}\}$, $\{in(*)_{Rec}\}$, $\{rand(*)_{tdpp}\}$, $\{rand(*)_{yval}\}$, $\{rand(*)_{cval1}\}$, $\{send(1,*)_{Trans}\}$, $\{send(2,*)_{Rec}\}$, $\{send(3,*)_{Trans}\}$, $\{out''(*)_{Rec}\}$, $\{fix-zval_{Rec}\}$, $\{fix-bval_{Trans}\}$.

State relation: q_1 and q_2 are related iff:
$q_1.inval(Trans) = \perp$ iff $q_2.inval(Trans) = \perp$, and similarly for $inval(Rec)$, $inval2(Trans)$, $tdpp$, $yval$, $zval$, $bval$, and $cval1$.

Figure 18: $TR2(D, Tdp)$, for the case where $C = \{Rec\}$.

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) $u.\mathit{TR1}.zval = s.Rec.zval$.

(i) $u.\mathit{TR1}.bval = s.Trans.bval$.

(j) $u.Src_{tdpp} = s.Src_{tdpp}$.

(k) $u.Src_{yval} = s.Src_{yval}$.

(l) $u.Adv' = s.Adv$.

(m) $u.Env = s.Env$.

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, for every S, T, $|corrtasks(S,T)| \leq 2$.*

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 $\{out'(x)_{Rec}\}$ 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 satisfies the two conditions in Lemma 3.55.
Start condition: It is obvious that the Dirac measures on execution fragments 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: We define $corrtasks : RS_{RS\|Env} \times RA_{RS\|Env} \to RA^*_{Int1\|Env}$ as follows:
For any $(S,T) \in (RS_{RS\|Env} \times RA_{RS\|Env})$:

- If $T \in \{\{in(x)_{Trans}\}, \{in(i)_{Rec}\}, \{choose-rand_{tdpp}\}, \{rand_{tdpp}\}, \{choose-rand_{yval}\}, \{rand_{yval}\}, \{fix-zval_{Rec}\}, \{send(1,f)_{Trans}\}, \{receive(1,f)_{Rec}\}, \{send(2,z)_{Rec}\}, \{receive(2,z)_{Trans}\}, \{send(3,b)_{Trans}\}, \{receive(3,b)_{Rec}\}$, or $\{out(x)_{Rec}\}\}$, then $corrtasks(S,T) = T$.
- If T is an output or internal task of *Env* or *Adv* that is not one of the tasks listed above, then $corrtasks(S,T) = T$.
- If $T = \{fix-bval_{Trans}\}$ then $corrtasks(S,T) = \{out'(x)_{Rec}\}\ \{fix-bval_{Trans}\}$.
- If $T = \{out'(x)_{Rec}\}$ then $corrtasks(S,T) = \{out''(x)_{Rec}\}$.

Suppose $(\epsilon_1, \epsilon_2) \in R$ and T is a task of $RS\|Env$ that is enabled in $supp(lstate(\epsilon_1))$. Let $\epsilon_1' = apply(\epsilon_1, T)$ and $\epsilon_2' = apply(\epsilon_2, corrtasks([lstate(\epsilon_1)], T))$.

The state equivalence property for ϵ_1 and ϵ_2 and Lemma 3.28 imply the state equivalence property for ϵ_1' and ϵ_2'; that is, there exist state equivalence classes $S_1 \in RS_{RS\|Env}$ and $S_2 \in RS_{Int1\|Env}$ such that $supp(lstate(\epsilon_1')) \subseteq S_1$ and $supp(lstate(\epsilon_2')) \subseteq S_2$.
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 1(m).

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 1(l).

Claim 2:

1. If T (defined above) is an output or internal task of Env, then
   (a) T is enabled in every state in $supp(lstate(\epsilon_2))$.
   (b) There is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   (c) 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
   (a) T is enabled in every state in $supp(lstate(\epsilon_2))$.
   (b) There is a unique action $a \in T$ that is enabled in every state in $supp(lstate(\epsilon_1)) \cup supp(lstate(\epsilon_2))$.
   (c) 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. The proof follows the same outline as for Lemma 9.7, except that instead of checking that Properties 1 and 2 are preserved, we need only check Property 1.

1. $T = \{in(x)_{Trans}\}$.

   Since T is an output task of Env, 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 $tr_{Env} = (q_{Env}, a, \mu_{Env})$ of Env from q_{Env} with action a. Here, $a = in(x)_{Trans}$ for a particular value of x.

   Next 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 execution fragments $\alpha \in supp(\epsilon'_1)$ 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 $s.Trans.inval \neq \bot$ then by Properties 1(a) and 1(d), $u.Funct.inval(Trans) \neq \bot$ and $u.TR1.inval2(Trans) \neq \bot$. 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 $s.Trans.inval = \bot$. Then again by Properties 1(a) and 1(d), $u.Funct.inval(Trans)$ $= u.TR1.inval2(Trans) = \bot$. 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 1(a) that $u.Funct.inval(Trans) = s.Trans.inval$, by 1(d) that $u.TR1.inval2(Trans) = s.Trans.inval$, and by 1(m) 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 1(a) and 1(d) hold for s' and u'. We also know that 1(m) holds by definition of ϵ'_{1j} and ϵ'_{2j}. Since no component other than $Trans.inval$ and Env in the RS system, and $Funct.inval(Trans)$, $TR1.inval2(Trans)$, 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}'.

2. $T = \{in(i)_{Rec}\}$.

   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.

   Since T is an output task of *Env*, 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 $tr_{Env} = (q_{Env}, a, \mu_{Env})$ of *Env* from q_{Env} with action a. Here, $a = in(i)_{Rec}$ for a particular value of i. Also, by next-transition determinism, it follows 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 execution fragments $\alpha \in supp(\epsilon_1')$ 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 $T = \{in(x)_{Trans}\}$. 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 1(b), 1(e), 1(l) and 1(m), instead of 1(a), 1(d) and 1(m).

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

   We first show that T is enabled in every state in $supp(lstate(\epsilon_2))$. Fix any state $u \in supp(lstate(\epsilon_2))$; we show that T 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 Src_{tdpp}, T is enabled in $s.Src_{tdpp}$. The precondition of T in the definition of Src_{tdpp} implies that $s.Src_{tdpp}.chosenval = \perp$. By Property 1(j), $u.Src_{tdpp} = s.Src_{tdpp}$. So, T is enabled in $u.Src_{tdpp}$, and hence in u, as needed.

   Next 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 execution fragments $\alpha \in supp(\epsilon_1')$ 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 1(j) 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}'.

4. $T = \{rand(p)_{tdpp}\}$.

We first show that T is enabled in every state in $supp(lstate(\epsilon_2))$. Fix any state $u \in supp(lstate(\epsilon_2))$; we show that T is enabled in u. Choose any $s \in supp(lstate(\epsilon_1))$. Since T is enabled in s and T is an output task of Src_{tdpp}, T is enabled in $s.Src_{tdpp}$ and $s.Src_{tdpp}.chosenval \neq \perp$. By Property 1(j), $u.Src_{tdpp} = s.Src_{tdpp}$. So, T is enabled in $u.Src_{tdpp}$, and hence in u, as needed.

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 1(j) 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 1(f) 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 1(f) 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.

5. $T = \{choose - rand_{yval}\}$.

   This case is analogous to the case for $T = \{choose - rand_{tdpp}\}$. In showing that T is enabled in every state in $supp(lstate(\epsilon_2))$, we use Property 1(k) instead of 1(j). In showing the step correspondence, we use the domain $\{0,1\} \to D$ instead of Tdp and also use Property 1(k) instead of 1(j).

6. $T = \{rand(y)_{yval}\}$.

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

   Since the application of T may cause probabilistic branching in *Adv*, to show the step correspondence, we proceed as for $T = \{in(x)_{Trans}\}$ but using *Adv* and *Adv'* instead of *Env*. In showing that Property 1 holds for ϵ'_{1j} and ϵ'_{2j}), for a fixed j, we use Properties 1(g) and 1(l) of R.

7. $T = \{fix - zval_{Rec}\}$.

   The fact that T is enabled in every state in $supp(lstate(\epsilon_2))$ follows from Properties 1(e), 1(g), 1(h), and 1(f) together with Lemma 6.4 5(b).

   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 trapdoor 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 6.4 5(b).

8. $T = \{fix - bval_{Trans}\}$.

   This is an interesting case, in which *bval* in the *RS* system is computed by *Trans* using its own input and the contents of a received round 2 message. It corresponds to two steps in the *Int1* system, in which *TR1* first receives a value from *Funct*, and then uses it in the computation of *bval* with the $fix-bval_{Trans}$ action.

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 Property 1(a) and 1(b), 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.

Now, let ϵ_2'' be the measure $apply(\epsilon_2, \{out'(x)_{Rec}\})$. We 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 1(f) applied to s and u. And $u.TR1.zval \neq \perp$, by Property 1(h) and Lemma 6.4 part 7(b). And $u.TR1.inval2(Trans) \neq \perp$, by Property 1(d). And $u.TR1.inval(Rec) \neq \perp$, by Lemma 6.4 parts 7b and 6 and Property 1(e). And finally, $u.TR1.bval = \perp$, by Property 1(i). 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 $u''.TR1.tdpp \neq \perp$, $u''.TR1.zval \neq \perp$, $u''.TR1.inval2(Trans) \neq \perp$, $u''.TR1.inval(Rec) \neq \perp$, 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 $(u'', fix-bval_{Trans}, \mu_u') \in D_{Int1\|Env}$. 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),

$$\begin{aligned}
&u'.TR1.bval(u.TR1.inval(Rec))\\
&\quad = B(u.TR1.yval(u.TR1.inval(Rec))) \oplus u''.TR1.inval(Trans)\\
&\quad = B(u.TR1.yval(u.TR1.inval(Rec))) \oplus u''.Funct.inval(Trans)(u''.Funct.inval(Rec))\\
&\quad = B(u.TR1.yval(u.TR1.inval(Rec))) \oplus u.Funct.inval(Trans)(u.Funct.inval(Rec))
\end{aligned}$$

Also, we have

$$\begin{aligned}
&u'.TR1.bval(1 - u.TR1.inval(Rec))\\
&\quad = B(u.TR1.tdpp.inverse(u.TR1.zval(1 - u.TR1.inval(Rec))))\\
&\qquad \oplus u.TR1.inval2(Trans)(1 - u.TR1.inval(Rec)).
\end{aligned}$$

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:

(a) $i = s.Rec.inval$

Then by Property 1(e), $i = u.TR1.inval(Rec)$. 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 1(a) and 1(b) for s and u. And $s.Trans.tdpp.inverse(s.Trans.zval(i))) = s.Rec.yval(i)$, by Lemma 6.4, part 10, which is equal to $u.TR1.yval(i))$ by Property 1(g). 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 $B(s.Trans.tdpp.inverse(s.Trans.zval(i))) \oplus s.Trans.inval(i) = B(u.TR1.yval(i)) \oplus u.Funct.inval(Trans)(u.Funct.inval(Rec))$.

(b) $i = 1 - s.Rec.inval$

Then $i = 1-u.TR1.inval(Rec)$ by Property 1(e). 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 1(d). And $s.Trans.tdpp = u.TR1.tdpp$ by Property 1(f). And $s.Trans.zval = u.TR1.zval$ by Property 1(h) and Lemma 6.4 part 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 1(c); 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 1(c) 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 6.4.
This in turn equals $u.Funct.inval(Trans)(u.Funct.inval(Rec))$ by Property 1(a) and 1(b) 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 1(c).

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.

9. $T = \{send(1, f)_{Trans}\}$.

We first show that T is enabled in every state in $supp(lstate(\epsilon_2))$. Fix any state $u \in supp(lstate(\epsilon_2))$; we show that T is enabled in u. Choose any $s \in supp(lstate(\epsilon_1))$. Since T is enabled in s and T is an output task of $Trans$, T is enabled in $s.Trans$, and so $s.Trans.tdpp \neq \perp$. By Property 1(f), $u.TR1.tdpp = s.Trans.tdpp$. So, T is enabled in $u.TR1$, and hence in u, as needed.

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 1(f) 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 1(l), $u.Adv' = s.Adv$. It is obvious that $u'.Adv' = s'.Adv$ and that 1(l) 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.

10. $T = \{send(2, z)_{Rec}\}$.

    We first show that T is enabled in every state in $supp(lstate(\epsilon_2))$. Fix any state $u \in supp(lstate(\epsilon_2))$; we show that T is enabled in u. Choose any $s \in supp(lstate(\epsilon_1))$. Since T is enabled in s and T is an output task of Rec, T is enabled in $s.Rec$, and therefore $s.Rec.zval \neq \perp$. By Property 1(h), $u.TR1.zval = s.Rec.zval \neq \perp$. So, T is enabled in $u.Rec$, and hence in u, as needed.

    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 1(h) 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 $T = \{send(1, f)_{Trans}\}$.

11. $T = \{send(3, b)_{Trans}\}$.

    The proof that T is enabled in every state in $supp(lstate(\epsilon_2))$ is analogous to the corresponding part of the proof for $T = \{send(1, f)_{Trans}\}$. Here we use Property 1(i), instead of 1(f).

    We also 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 $T = \{send(1, f)_{Trans}\}$.

12. $T = \{receive(1, f)_{Rec}\}$.

    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, a is $receive(1, f)_{Rec}$ for a fixed value of f.

    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 use the fact that application of T updates only $Rec.tdp$ in RS and that R does not depend on this component.

13. $T = \{receive(2, z)_{Trans}\}$.

    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 a is $receive(2, z)_{Trans}$ for a fixed value of 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 make use of the fact that the application of T updates *Trans.zval* only, which has no effect on R.

14. $T = \{receive(3, b)_{Rec}\}$.

    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 a is $receive(3, b)_{Rec}$ for a fixed value of 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 Property 1(c) is 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 $(u, a, \mu_u) \in D_{Int1\|Env}$.

    We know that $s'.Rec.outval \neq \perp$. Then $s'.Rec.outval = s'.Trans.inval(s'.Rec.inval)$ by Lemma 6.4, which is equal to $s.Trans.inval(s.Rec.inval)$.
    This in turn equals $u.Funct.inval(Trans)(u.Funct.inval(Rec))$ by Property 1(a) and 1(b) for s and u. Now, $s.Trans.bval \neq \perp$, by Lemma 6.4, part 4, so by Property 1(i), $u.TR1.bval \neq \perp$. 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 1(c), as needed.

15. $T = \{out(x)_{Rec}\}$.

    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 $T = \{in(x)_{Trans}\}$, decomposing the measures generated by the application of T according to the resulting state in *Env*, and using Property 1(m) 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}.

16. $T = \{out'(x)_{Rec}\}$.

    We first show that the corresponding task $\{out''(x)\}$ is enabled in every state in $supp(lstate(\epsilon_2))$. Fix any state $u \in supp(lstate(\epsilon_2))$; we show that $\{out''(x)_{Rec}\}$ is enabled in u. Note that $\{out''(x)_{Rec}\}$ is an output task of *TR1* in the *Int1* system. Choose any $s \in supp(lstate(\epsilon_1))$. Since T is enabled in s and 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 Property 1(c), $u.TR1.inval(Trans) \neq \perp$. So, $\{out''(x)_{Rec}\}$ is enabled in $u.TR1$, and hence in u, as needed.

    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.

17. T is an output task of *Env* and an input task of *Adv*.

Since T is an output task of *Env*, 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 $tr_{Env} = (q_{Env}, a, \mu_{Env})$ of *Env* from q_{Env} with action a. Also, by next-transition determinism, it follows 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 execution fragments $\alpha \in supp(\epsilon'_1)$ 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 $T = \{in(x)_{Trans}\}$. 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 1(l) and 1(m).

18. T is either an output task of *Env* that is not an input task of *Adv*, *Trans*, or *Rec*, or is an internal task of *Env*.

    Since T is an output or internal task of *Env*, 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 $tr_{Env} = (q_{Env}, a, \mu_{Env})$ of *Env* from q_{Env} with action a.

    To show the step correspondence, we proceed as for $T = \{in(x)_{Trans}\}$. 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 1(m).

    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}.

19. T is an output task of *Adv* and an input task of *Env*.

    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. 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 $T = \{in(x)_{Trans}\}$, using Properties 1(l) and 1(m).

    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}.

20. T is either an output task of *Adv* that is not an input task of *Env*, *Trans*, or *Rec*, or is an internal task of *Adv*.

    Since T is an output or internal 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.

    To show the step correspondence, we proceed as for $T = \{in(x)_{Trans}\}$, 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 1(l).

    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}.

$\square$

Proof. (Of Lemma 10.4:)
By Lemma 10.6, R is a simulation relation from $RS_k \| Env$ to $Int1_k \| Env$. Then Theorem 3.53 implies that $tdists(RS_k \| Env) \subseteq tdists(Int1_k \| Env)$. Since Env was chosen arbitrarily, this implies (by definition of $\leq_0$) that $RS_k \leq_0 Int1_k$. $\square$

Proof. (Of Lemma 10.5:)
By Lemma 10.6, R is a simulation relation from $RS_k \| Env$ to $Int1_k \| Env$ for which $|corrtasks(S,T)| \leq 2$ for every S and T. Since that lemma holds for every k and every Env, Theorem 3.86 implies that $\overline{RS} \leq_{neg,pt} \overline{Int1}$. $\square$

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:

- $\overline{SInt1} = hide_{\{rand(*)_{tdpp}\} \cup \{rand(*)_{zval}\}}(\overline{TR1} \| \overline{Src_{tdpp}} \| \overline{Src_{zval}})$,
- $\overline{SInt2} = hide_{\{rand(*)_{tdpp}\} \cup \{rand(*)_{zval}\} \cup \{rand(*)_{cval1}\}}(\overline{TR2} \| \overline{Src_{tdpp}} \| \overline{Src_{zval}} \| \overline{Src_{cval1}})$.

Next, using mappings of the sort we used in Section 9.4, we will show that $\overline{SInt1} \leq_0 \overline{SHOT'}$ and $\overline{SHROT'} \leq_0 \overline{SInt2}$, where $\overline{SHOT'}$ and $\overline{SHROT'}$ are the families defined in Section 8.3.3. More precisely, we prove that $SInt1_k \leq_0 SHOT'_k$ and $SHROT'_k \leq_0 SInt2_k$ for every k. In the rest of this subsection, we suppress the mention of k everywhere.

Finally, using the properties of these mappings and the different properties of the $\leq_{neg,pt}$ relation, we prove the expected relation.

10.5.1 The *SInt1* subsystem implements *SHOT'*

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 execution fragments 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 $u.Ifc'.fval = s.TR1.tdpp.funct$.
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. if $s.TR1.yval \neq \bot$ then $u.H.yval \neq \bot$ and $u.Ifc'.yval' \neq \bot$.
10. if $s.TR1.zval = \bot$ then $u.Ifc'.zval = \bot$ 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 $s.TR1.bval = \bot$ then $u.Ifc'.bval = \bot$ 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. $u.Env' = s.Env'$.

Lemma 10.8 *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 five steps of $SHOT'\|Env'$, that is, for every S, T, $|corrtasks(S,T)| \leq 5$.*

Proof. We prove that R satisfies the two conditions in Lemma 3.55.
Start condition: It is obvious that the Dirac measures on execution fragments 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 $\bot$.
Step condition: We define $corrtasks(RS_{SInt1\|Env'} \times RA_{SInt1\|Env'}) \rightarrow RA^*_{SHOT'\|Env'}$ as follows:
For any $(S,T) \in (RS_{SInt1\|Env'} \times RA_{SInt1\|Env'})$:

- If $T \in \{\{in(x)_{Trans}\}, \{out'(x)_{Rec}\}, \{out''(x)_{Rec}\}, \{in(i)_{Rec}\}, \{send(1,f)_{Trans}\}, \{send(2,z)_{Rec}\}, \{send(3,b)_{Trans}\}\}$ then $corrtasks(S,T) = T$.
- If T is an output or internal task of Env' that is not one of the tasks listed above, then $corrtasks(S,T) = T$.
- If $T = \{choose-rand_{tdpp}\}$ then $corrtasks(S,T) = \{choose-rand_{tdp}\}$.
- If $T = \{choose-rand_{yval}\}$ then $corrtasks(S,T) = \{choose-rand_{yval}\}\{choose-rand_{yval'}\}$.
- If $T = \{rand(p)_{tdpp}\}$ then $corrtasks(S,T) = \{rand(f)_{tdp}\}$.
- If $T = \{rand(y)_{yval}\}$ then $corrtasks(S,T) = \{rand(y)_{yval}\}\{rand(y)_{yval'}\}$.
- If $T = \{fix{-}zval_{Rec}\}$ then $corrtasks(S,T) = \{fix{-}zval\}\{rand(z)_{zval}\}\{fix{-}bval\}\{rand(b)_{bval}\}\{fix-zval_{Rec}\}$.
- If $T = \{fix-bval_{Trans}\}$ then $corrtasks(S,T) = \{fix-bval_{Trans}\}$.

Suppose $(\epsilon_1, \epsilon_2) \in R$ and T is a task of $SInt1\|Env'$ that is enabled in $supp(lstate(\epsilon_1))$. equivalence follows as in that proof. Identical versions of Claim 1 and Claim 2 in that proof carry over for Env' to this case. We simply verify that the tasks in $corrtasks(S,T)$ are enabled when T is enabled: the other aspects of the proof are similar to the corresponding ones in Lemma 10.6.

1. $T \in \{\{in(x)_{Trans}\}, \{out'(x)_{Rec}\}, \{in(i)_{Rec}\}\}$. In these cases, T is an input task of $SInt1$, which is also the case of $corrtasks(S,T) = T$ in $SHOT'$. These input tasks are always enabled.
2. $T = \{out''(x)_{Rec}\}$. Consider any states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$. Since T is enabled in s, we know that $s.TR1.inval(Trans) \neq \bot$. Now, since $\epsilon_1 R \epsilon_2$, we know that $u.Ifc'.inval(Trans) \neq \bot$. This is sufficient to have T in enabled in u.
3. $T = \{send(1,f)_{Trans}\}$. This case is similar to the previous one since we know that $s.TR1.tdpp \neq \bot$ and $u.Ifc.fval = s.TR1.tdpp.funct$ in any states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$.

4. $T = \{send(2, z)_{Rec}\}$. Again, this case is similar to the previous one since we know that $s.TR1.zval \neq \perp$, which implies that $u.Ifc'.zval \neq \perp$ in any states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$.

5. $T = \{send(3, b)_{Trans}\}$. Again, this case is similar to the previous one since we know that $s.TR1.bval \neq \perp$, which implies that $u.Ifc'.bval \neq \perp$ in any states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$.

6. T is an output or internal task of Env' that is not one of the tasks listed above. Consider any states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$. Since T is enabled in s, it is also enabled in u since we know that $s.Env' = u.Env'$.

7. $T = \{choose - rand_{tdpp}\}$. We know that $\{choose - rand_{tdp}\}$ is enabled in $SHOT'$ because $u.Src_{tdp}.chosenval = \perp$ when $s.Src_{tdpp}.chosenval = \perp$ in any states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$.

8. $T = \{choose{-}rand_{yval}\}$. We know that $\{choose{-}rand_{yval}\}$ and $\{choose{-}rand_{yval'}\}$ are enabled in $SHOT'$ because $u.Src_{yval}.chosenval = u.Src_{yval'}.chosenval = \perp$ when $s.Src_{yval}.chosenval = \perp$ in any states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$.

9. $T = \{fix - zval_{Rec}\}$. Consider any states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$. Since T is enabled in s, we know that $s.TR1.yval \neq \perp$ and $s.TR1.tdpp \neq \perp$. Since $\epsilon_1 R \epsilon_2$, we also know that $u.H.yval \neq \perp$, $u.Ifc''.yval' \neq \perp$ and $u.Ifc'.fval = u.H.fval \neq \perp$. So, the sequence of tasks $\{fix - zval\}\{rand(z)_{zval}\}\{fix - bval\}\{rand(b)_{bval}\}$ is enabled in u. After these tasks have been performed, $u.Ifc'.zval' \neq \perp$ and $u.Ifc'.bval' \neq \perp$. Now, since T is enabled in s, we know that $s.TR1.inval(Rec) \neq \perp$ and $s.TR1.zval = \perp$. Since $\epsilon_1 R \epsilon_2$, we also know that $u.Ifc'.inval(Rec) \neq \perp$ and $u.Ifc'.zval = \perp$. So, at this point, the $\{fix - zval_{Rec}\}$ task is enabled.

10. $T = \{fix - bval_{Trans}\}$. Consider any states $s \in supp(lstate(\epsilon_1))$ and $u \in supp(lstate(\epsilon_2))$. Since T is enabled in s and $\epsilon_1 R \epsilon_2$, we know that

    - $s.TR1.zval \neq \perp$, which implies that $u.Ifc'.zval \neq \perp$,
    - $s.TR1.inval(Trans) \neq \perp$, which implies that $u.Ifc'.inval(Trans) \neq \perp$,
    - $s.TR1.inval2(Trans) \neq \perp$, which implies that $u.Ifc'.inval2(Trans) \neq \perp$,
    - $s.TR1.inval(Rec) \neq \perp$, which implies that $u.Ifc'.inval(Rec) \neq \perp$,
    - $s.TR1.bval = \perp$, which implies that $u.Ifc'.bval = \perp$.

    Now, we observe that, if $u.Ifc'.zval \neq \perp$, then $u.Ifc'.yval' \neq \perp$ and $u.Ifc'.bval' \neq \perp$. So, all preconditions of the $\{fix - bval_{Trans}\}$-task are verified in u.

□

10.5.2 $SHROT'$ 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 execution fragments 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 $u.TR2.tdpp.funct = s.Ifc'.fval$.

7. if $s.Src_{zval}.chosenval = \perp$ then $u.Src_{yval}.chosenval = \perp$.

8. if $s.Src_{zval}.chosenval \neq \perp$ then $lstate(\epsilon_2).Src_{yval}.chosenval$ is the uniform distribution on $(\{0,1\} \to D)$.

9. if $s.Ifc'.zval' \neq \perp$ then $u.TR2.yval \neq \perp$.

10. 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
    - $u.TR2.zval(1 - u.TR2.inval(Rec)) = s.Ifc'.zval(1 - s.Ifc'.inval(Rec))$.

11. 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
    - $u.TR2.bval(1 - u.TR2.inval(Rec)) = s.Ifc'.bval(1 - s.Ifc'.inval(Rec))$.

12. $u.Env' = s.Env'$.

Lemma 10.9 *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, for every S, T, $|corrtasks(S,T)| \leq 1$.*

Proof. We show that R satisfies the two conditions in Lemma 3.55.
Start condition: It is obvious that the Dirac measures on execution fragments 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: We define $corrtasks(RS_{SHROT' \| Env'} \times RA_{SHROT' \| Env'}) \to RA^*_{SInt2 \| Env'}$ as follows:
For any $(S,T) \in (RS_{SHROT' \| Env'} \times RA_{SHROT' \| Env'})$:

- If $T \in \{\{in(x)_{Trans}\}, \{out'(x)_{Rec}\}, \{out''(x)_{Rec}\}, \{in(i)_{Rec}\}, \{fix-zval_{Rec}\}, \{fix-bval_{Trans}\}, \{send(1,f)_{Trans}\}, \{send(2,z)_{Rec}\}, \{send(3,b)_{Trans}\}\}$ then $corrtasks(S,T) = \{T\}$.
- If T is an output or internal task of Env' that is not one of the tasks listed above, then $corrtasks(S,T) = T$.
- If $T = \{choose-rand_{tdp}\}$ then $corrtasks(S,T) = \{choose-rand_{tdpp}\}$.
- If $T = \{rand(f)_{tdp}\}$ then $corrtasks(S,T) = \{rand(p)_{tdpp}\}$.
- If $T = \{choose-rand_{yval'}\}$ then $corrtasks(S,T) = \lambda$.
- If $T = \{rand(y)_{yval'}\}$ then $corrtasks(S,T) = \lambda$.
- If $T = \{choose-rand_{zval}\}$ then $corrtasks(S,T) = \{choose-rand_{yval}\}$.
- If $T = \{rand(z)_{zval}\}$ then $corrtasks(S,T) = \{rand(y)_{yval}\}$.
- If $T = \{choose-rand_{bval}\}$ then $corrtasks(S,T) = \{choose-rand_{cval1}\}$.
- If $T = \{rand(b)_{bval}\}$ then $corrtasks(S,T) = \{rand(c)_{cval1}\}$.

The only interesting cases in this mapping are those corresponding to the selection and to the transmission of $s.Src_{yval'}.chosenval$ and $s.Src_{zval}.chosenval$ (for any state $s \in supp(lstate(\epsilon_1))$). These two values are selected into two random sources in $SHROT'$ while they are both selected into the Src_{yval} random source in $SInt2$.

Since all actions of Ifc' require that both these values are defined (or do not care about them), we manage these differences in a simple way: we do not define any task corresponding to the tasks of the $Src_{yval'}$ source, and make the tasks of the Src_{zval} automata correspond. This is sufficient to be sure that $TR2.yval \neq \perp$ when both $Ifc'.yval'$ and $Ifc'.zval'$ have been set.

Proving the rest of this correspondence is fairly obvious.

□

10.5.3 *Int1* implements *Int2*

Proof. (of Lemma 10.7)
In Lemma 10.8 and 10.9, 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. 3.86 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 8.16) and since the $\leq_{neg,pt}$ implementation relation is transitive (see Lemma 3.83), 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 3.84, 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:

$$\begin{aligned} &hide_{\{rand(*)_{tdpp}\} \cup \{rand(*)_{zval}\}}(\overline{Funct} \| \overline{Adv} \| \overline{TR1} \| \overline{Src_{tdpp}} \| \overline{Src_{zval}}) \\ &\quad \leq_{neg,pt} hide_{\{rand(*)_{tdpp}\} \cup \{rand(*)_{zval}\} \cup \{rand(*)_{cval1}\}}(\overline{Funct} \| \overline{Adv} \| \overline{TR2} \| \overline{Src_{tdpp}} \| \overline{Src_{zval}} \| \overline{Src_{cval1}}) \end{aligned}$$

or, in other words, $\overline{Int1} \leq_{neg,pt} \overline{Int2}$, as needed. □

10.6 *Int2* implements *SIS*

We show:

Lemma 10.10 *For every k, $Int2_k \leq_0 SIS_k$.*

We prove Lemma 9.11 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 3.53, 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 10.11 $\overline{Int2} \leq_{neg,pt} \overline{SIS}$.

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 execution fragments of *Int2* and *SIS*, respectively, satisfying the following properties:

1. **Trace distribution equivalence:** $tdist(\epsilon_1) = tdist(\epsilon_2)$.
2. **State equivalence:** There exist state equivalence classes $S_1 \in RS_{Int2\|Env}$ and $S_2 \in RS_{SIS\|Env}$ such that $supp(lstate(\epsilon_1)) \subseteq S_1$ and $supp(lstate(\epsilon_2)) \subseteq S_2$.

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 $u.TR.bval1 \neq \perp$ 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) $u.Src_{bval1}.chosenval = TR2.bval$.
   (m) $u.Adv' = s.Adv'$.
   (n) $u.Env = s.Env$.
2. For every $u \in supp(lstate(\epsilon_2))$: If $u.TR.bval1 = \perp$ then one of the following holds:
   (a) For every $s \in supp(lstate(\epsilon_1))$, $s.Src_{cval1}.chosenval = \perp$.
   That is, $cval1$ has not been chosen.
   (b) For every $s \in supp(lstate(\epsilon_1))$, $s.TR2.cval1 = \perp$, and $lstate(\epsilon_1)$ projected on $Src_{cval1}.chosenval$ is the uniform distribution on $\{0, 1\}$.
   (c) $lstate(\epsilon_1)$ projected on $TR2.cval1$ is the uniform distribution on $\{0, 1\}$.

10.6.2 The mapping proof

Lemma 10.12 *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, for every S, T, $|corrtasks(S, T)| \leq 3$.*

Proof. We prove that R satisfies the two conditions in Lemma 3.55.
Start condition: It is obvious that the Dirac measures on execution fragments 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 $s.Src_{cval1}.chosenval = \perp$.

Step condition: We define $corrtasks : RS_{Int2\|Env} \times RA_{Int2\|Env} \to RA^*_{SIS\|Env}$ as follows:
For any $(S, T) \in RS_{Int2\|Env} \times RA_{Int2\|Env}$:

- If $T \in \{\{in(x)_{Trans}\}, \{in(i)_{Rec}\}, \{choose{-}rand_{tdpp}\}, \{rand_{tdpp}\}, \{choose{-}rand_{zval}\}, \{rand_{zval}\}, \{send(1,f)_{Trans}\}, \{receive(1,f)_{Rec}\}, \{send(2,z)_{Rec}\}, \{receive(2,z)_{Trans}\}, \{send(3,b)_{Trans}\}, \{receive(3,b)_{Rec}\}$, or $\{out(x)_{Rec}\}\}$, then $corrtasks(S,T) = T$.

- If T is an output or internal task of *Env* or *Adv* that is not one of the tasks listed above, then $corrtasks(S,T) = T$.

- If $T \in \{\{choose - rand_{cval1}\}, \{rand_{cval1}\}\}$ then $corrtasks(S,T) = \lambda$.

- If $T = \{fix - bval_{Trans}\}$ then $corrtasks(S,T) = \{choose - rand_{bval1}\}\ \{rand_{bval1}\}\ \{fix - bval_{Trans}\}$.

Suppose $(\epsilon_1, \epsilon_2) \in R$ and T is a task of $Int2\|Env$ that is enabled in $supp(lstate(\epsilon_1))$. Let $\epsilon_1' = apply(\epsilon_1, T)$ and $\epsilon_2' = apply(\epsilon_2, corrtasks([lstate(\epsilon_1)], T))$.
We establish the step condition by considering cases based on the value of T. The proof follows the same outline as for Lemma 9.7.

1. $T = \{in(x)_{Trans}\}$.

   Task T is output from *Env* to both *Funct* and *TR2* in the *Int2* system, and from *Env* to *Funct* in the *SIS* system.

   Since T is an output task of *Env*, 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 $tr_{Env} = (q_{Env}, a, \mu_{Env})$ of *Env* from q_{Env} with action a. Here, $a = in(x)_{Trans}$ for a particular value of x.

   Next, 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 execution fragments $\alpha \in supp(\epsilon_1')$ 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 $j \in I$; we show that $(\epsilon_{1j}', \epsilon_{2j}') \in R$. To do this, we establish Properties 1 and 2 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 $(u, a, \mu_u) \in D_{SIS\|Env}$.

   If $s.TR2.inval2(Trans) \neq \bot$ then by Properties 1(a) and 1(b), $u.Funct.inval(Trans) \neq \bot$ and $s.Funct.inval(Trans) \neq \bot$. 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'.

   Now suppose that $s.TR2.inval2(Trans) = \bot$. Then again by Properties 1(a) and 1(b), $u.Funct.inval(Trans) = s.Funct.inval(Trans) = \bot$. Then by the definitions of *Int2* and *SIS*, we know that application of T updates $TR2.inval2(Trans)$ and $Funct.inval(Trans)$ in *Int2*, and $Funct.inval(Trans)$ in *SIS*. It also updates the state of *Env* to q_j in both systems.

   We know by Property 1(a) that $u.Funct = s.Funct$, by Property 1(b) that $u.Funct.inval(Trans) = s.TR2.inval2(Trans)$, and by 1(n) that $u.Env = s.Env$. By the effects of T in definitions of *Funct* and *TR2*, we know that $u'.Funct = s'.Funct$ and $u'.Funct.inval(Trans) = s'.TR2.inval2(Trans)$; hence, Properties 1(a) and 1(b) hold for s' and u'. We also know that 1(n) holds for s' and u' by definition of ϵ_{1j}' and ϵ_{2j}': in both s' and u', the state of *Env* is q_j. Since no state component other than $TR2.inval2$, $Funct.inval(Trans)$, and *Env* in the $TRtwo$ system, and $Funct.inval(Trans)$ and *Env* 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'.

The proof of Property 2 is analogous to the corresponding proof in Lemma 9.13.

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. $T = \{in(i)_{Rec}\}$.

   Task T is output from *Env* to *Funct*, *Adv′* and *TR2* in the *Int2* system, and from *Env* to *Funct*, *Adv′* and *TR* in the *SIS* system.

   Since T is an output task of *Env*, 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 $tr_{Env} = (q_{Env}, a, \mu_{Env})$ of *Env* from q_{Env} with action a. Here, $a = in(i)_{Rec}$ for a particular value of i.

   The rest of the proof for this case follows the proof for $T = \{in(x)_{Trans}\}$. The only difference is that, in showing that Property 1 holds for ϵ'_{1j} and ϵ'_{2j}, for a fixed j, we use the fact that application of T affects only *Funct.inval*(*Rec*), *Env*, the "new" state components of *Adv′*, and *TR2.inval*(*Rec*) in the *Int2* system, and *Funct.inval*(*Rec*), *Env*, the "new" state components of *Adv′*, and *TR.inval*(*Rec*) in the *SIS* system. We use Properties 1(a), 1(d), 1(m), and 1(n).

   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}.

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

   Identical to the corresponding case in the proof of Lemma 10.6, using Property 1(j).

4. $T = \{rand(p)_{tdpp}\}$.

   Identical to the corresponding case in the proof of Lemma 10.6, using Properties 1(e) and 1(j).

5. $T = \{choose-rand_{yval}\}$.

   Identical to the corresponding proof case in the proof of Lemma 10.6, using Property 1(k).

6. $T = \{rand(y)_{yval}\}$.

   Enabling is shown by using 1(k) and the resulting distributions are related by using 1(f).

7. $T = \{choose-rand_{cval1}\}$.

   We know that for all states in $supp(lstate(\epsilon_1))$, $cval1$ has not yet been chosen.
   That is, $Src_{cval1}.chosenval = \bot$. Now, applying T to ϵ_1 gives ϵ'_1 such that $\epsilon'_1.Src_{cval1}.chosenval$ is the uniform distribution on $\{0, 1\}$. Since applying λ yields $\epsilon'_2 = \epsilon_2$, we can use 2(b) to show that $(\epsilon'_1, \epsilon'_2) \in R$.

8. $T = \{rand(*)_{cval1}\}$.

   We know that for all states in $supp(lstate(\epsilon_1))$, $cval1$ has already been chosen.
   That is, $Src_{cval1}.chosenval \neq \bot$. Let $\epsilon'_1 = apply((,\epsilon)_1, T)$. We know that all states in $supp(lstate(\epsilon'_1))$, $TR2.cval1 \neq \bot$ and $lstate(\epsilon'_1).Src_{cval1}$ is the uniform distribution on $\{0, 1\}$. Applying λ yields $\epsilon'_2 = \epsilon_2$.

   Let p be the Dirac measure on the singleton index set $\{1\}$. Then, the only interesting part of the proof is showing that $(\epsilon'_1, \epsilon'_2) \in R$. To show this, we use Property 2(b) of R.

9. $T = \{out'(x)_{Rec}\}$.

   T is output from *Funct* to *TR2* in the *Int2* system and from *Funct* to *TR* in the *SIS* system.

   We show the enabling of $\{out'(x)_{Rec}\}$ in all states in $supp(lstate(\epsilon_2))$ by using Property 1(a). To see that $(\epsilon'_1, \epsilon'_2) \in R$, we use Property 1(c).

10. $T = \{fix - zval_{Rec}\}$.

    The fact that T is enabled in all states in $supp(lstate(\epsilon_2))$ follows from Properties 1(f), 1(d), 1(e) and 1(g). To see that $(\epsilon_1', \epsilon_2') \in R$, we use Property 1(g). This is straightforward because $zval$ is computed in the same way in $TR2$ and TR.

11. $T = \{fix - bval_{Trans}\}$.

    Here, a deterministic step in the $Int2$ system maps to a random choice followed by two deterministic steps in the SIS system.

    We first show that the sequence of tasks $\{choose - rand_{bval1}\}\ \{rand(b)_{bval1}\}\ \{fix - bval_{Trans}\}$ is enabled in $supp(lstate(\epsilon_2))$.

    Since T is enabled in every state $s \in supp(lstate(\epsilon_1))$, we know that $s.TR2.yval$, $s.TR2.cval1$, $s.TRtwo.inval(Trans)$, $s.TR2.inval2(Trans)$, and $s.TR2.inval(Rec) \neq \perp$, and $s.TR2.bval = \perp$ in every state $s \in supp(lstate(\epsilon_1))$. Then by Property 1, we know that $u.TR.yval \neq \perp$ (by 1(f)), $u.TR.inval(Trans) \neq \perp$ (by 1(c)), $u.TR.inval(Rec) \neq \perp$ (by 1(d)), and $u.TR.bval = \perp$ (by 1(i)). Then by Property 1(k), we know that $u.Src_{bval1}.chosenval = \perp$. Therefore, $\{choose - rand_{bval1}\}$ is enabled from all states in $supp(lstate(\epsilon_2))$.

    Let $\epsilon_2'' = apply(\epsilon_2, \{choose - rand_{bval1}\})$. Clearly, $\{rand(*)_{bval1}\}$ is enabled from all states in $supp(lstate(\epsilon_2''))$.

    Let $\epsilon_2''' = apply(\epsilon_2'', \{rand_{bval1}\})$. Then we claim that $\{fix - bval_{Trans}\}$ is enabled from all states in $supp(\epsilon_2''')$. Let $u''' \in supp(\epsilon_2''')$. Then by the effects of the first two tasks in the sequence, we see that $u''.TR.yval \neq \perp$, $u''.TR.inval(Trans) \neq \perp$, $u''.TR.inval(Rec) \neq \perp$, and $u''.TR.bval = \perp$. Also, by the effects of $\{rand(b)_{bval1}\}$, we have that $u''.TR.bval1 \neq \perp$. Since these are all the preconditions for $fix - bval_{Trans}$ in TR, we have that $\{fix - bval_{Trans}\}$ is enabled from u''', as needed.

    To see that $(\epsilon_1', \epsilon_2') \in R$, we use Property 1(h).

12. $T = \{out''(x)_{Rec}\}$.

    T is output from $TR2$ to Adv' in the $Int2$ system and from TR to Adv' in the SIS system. Enabling follows from 1(c) and we can show that ϵ_1' and ϵ_2' are related by using 1(m).

13. $T = \{send(1, f)_{Trans}\}$.

    Identical to the corresponding case in the proof of Lemma 10.6, except that here we replace *Trans* with $TR2$ and $TR1$ with TR and use Properties 1(e) and 1(m).

14. $T = \{send(2, z)_{Rec}\}$.

    Identical to the corresponding case in the proof of Lemma 10.6, except that here we replace *Rec* with $TR2$ and $TR1$ with TR, and use Property 1(g).

15. $T = \{send(3, b)_{Trans}\}$.

    Identical to the corresponding case in the proof of Lemma 10.6, except that here we replace *Trans* with $TR2$ and $TR1$ with TR, and use Property 1(i).

16. $T = \{receive(1, f)_{Rec}\}$.

    Identical to the corresponding case in the proof of Lemma 10.6, except that here we replace *Rec* with $TR2$. In showing Property 1, we use the fact that applying T has no effect in either system.

17. $T = \{receive(2, z)_{Trans}\}$.

    Identical to the corresponding case in the proof of Lemma 10.6, except that here we replace *Trans* with $TR2$. In showing Property 1, we use the fact that applying T has no effect in either system.

18. $T = \{receive(3, b)_{Rec}\}$.

    Identical to the corresponding case in the proof of Lemma 10.6, except that here we replace *Rec* with *TR2*. In showing Property 1, we use the fact that applying T has no effect in either system.

19. $T = \{out(x)_{Rec}\}$.

    This is output from from Adv' to *Env* in both systems. We use Claim 2 to show enabling. The only interesting aspect of this proof is that *Env* may make a probabilistic choice on the application of T. The step correspondence can be shown by decomposing the distributions generated by application of T as in the case for $T = \{in(x)_{Trans}\}$.

20. T is an output task of *Env* and an input task of *Adv*.

    Identical to the corresponding case in the proof of Lemma 10.6.

21. T is an output task of *Env* that is not an input task of *Adv*, *Funct*, or *TR2*, or T is an internal task of *Env*.

    Identical to the corresponding case in the proof of Lemma 10.6.

22. T is an output task of *Adv* and an input task of *Env*.

    Identical to the corresponding case in the proof of Lemma 10.6.

23. T is an output task of *Adv* that is not an input task of *Env*, *Funct*, or *TR2*, and is not a *receive* task, or else T is an internal task of *Adv*.

    Identical to the corresponding case in the proof of Lemma 10.6.

$\square$

Proof. (Of Lemma 10.10:)
By Lemma 10.12, R is a simulation relation from $Int2_k \| Env$ to $SIS_k \| Env$. Then Theorem 3.53 implies that $tdists(Int2_k \| Env) \subseteq tdists(SIS_k \| Env)$. Since *Env* was chosen arbitrarily, this implies (by definition of $\leq_0$) that $RS_k \leq_0 Int1_k$. $\square$

Proof. (Of Lemma 10.11:)
By Lemma 10.12, R is a simulation relation from $Int2_k \| Env$ to $SIS_k \| Env$ for which $|corrtasks(S, T)| \leq 3$ for every S and T. Since that lemma holds for every k and every *Env*, Theorem 3.86 implies that $\overline{Int2} \leq_{neg,pt} \overline{SIS}$. $\square$

10.7 Putting the pieces together

Proof. (of Theorem 10.1):
Lemmas 10.5, 10.7, and 10.11, and transitivity of $\leq_{neg,pt}$, imply that $\overline{RS} \leq_{neg,pt} \overline{SIS}$. Since the simulator $SSim_k$ satisfies the constraints for a simulator in Figure 2, this implies that $\overline{RS} \leq_{neg,pt} \overline{IS}$. $\square$

11 Correctness Proof, Case 3: Transmitter Corrupted

Next, we consider the case where only the transmitter is corrupted. We prove the following theorem:

Theorem 11.1 *Let $\overline{RS}$ be a real-system family for $(\overline{D}, \overline{Tdp}, C)$, $C = \{Trans\}$, in which the family $\overline{Adv}$ of adversary automata is polynomial-time-bounded.*
Then there exists an ideal-system family $\overline{IS}$ for $C = \{Trans\}$, in which the family $\overline{Sim}$ is polynomial-time-bounded, and such that $\overline{RS} \leq_{neg,pt} \overline{IS}$.

Again, we drop explicit mention of C. Again, we express each Sim_k as a composition of automata, and show that $\overline{RS}$, the real-system family, implements the (new) structured-ideal-system family $\overline{SIS}$. This time, we do not need intermediate levels, because we do not need a Distinguisher argument.

11.1 Simulator structure

For each k, we define a structured simulator $SSim_k$, as the composition of five task-PIOAs:

- $Trans(D_k, Tdp_k)$, as in RS_k.
- $(Src(Tdpp_k)_{tdpp})_k$, isomorphic to $Src(Tdpp_k)$.
- $(Src(\{0,1\} \rightarrow D_k)_{zval})_k$, isomorphic to $Src(\{0,1\} \rightarrow D_k)$
- $(RecSim(D_k))_k$, an abstract version of *Rec*.
- Adv_k, as in RS_k.

Trans is connected to *Adv* as in the real system. *RecSim* has *send* outputs that are inputs to *Adv*, but has no *receive* inputs. *Adv* also has $in(x)_{Trans}$ inputs, which come from *Env*. The outputs of Src_{tdpp} go both to *Trans* and to *Adv*. The outputs of Src_{zval} go to *RecSim* only.

$RecSim(D)$ is defined in Figure 19. It simply chooses a pair of D values at random and sends it in round 2 messages.

$RecSim(D)$, where $C = \{T\}$:

Signature:

Input:
$rand(z)_{zval}, z \in (\{0,1\} \rightarrow D)$

Output:
$send(2,z)_{Rec}$, $z \in (\{0,1\} \rightarrow D)$

State:
$zval \in (\{0,1\} \rightarrow D) \cup \{\perp\}$, initially $\perp$

Transitions:

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

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

Tasks: $\{rand(*)_{zval}\}$, $\{send(2,*)_{Rec}\}$.

State relation: q_1 and q_2 are related iff:
$q_1.zval = \perp$ iff $q_2.zval = \perp$.

Figure 19: Code for $RecSim(D)$, where $C = \{Trans\}$.

We define SIS_k, the structured ideal system, to be $Funct_k \| SSim_k$. We show:

Lemma 11.2 *For every k, $RS_k \leq_0 SIS_k$.*

Lemma 11.3 $\overline{RS} \leq_{neg,pt} \overline{Int1}$.

In the rest of this subsection, we fix *Env*, an environment for RS_k and SIS_k. We suppress mention of k.

11.2 State correspondence

Here we define the correspondence R from states of $RS \| Env$ to states of $SIS \| Env$, which we will show to be a simulation relation in Section 11.3.

Let ϵ_1 and ϵ_2 be discrete probability measures on finite execution fragments of $RS \| Env$ and $SIS \| Env$, respectively, satisfying the following properties:

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

2. **State equivalence:** There exists state equivalence classes $S_1 \in RS_{RS\|Env}$ and $S_2 \in RS_{Int1\|Env}$ such that $supp(lstate(\epsilon_1)) \subseteq S_1$ and $supp(lstate(\epsilon_2)) \subseteq S_2$.

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) $u.Trans = s.Trans$.

   (d) $u.Src_{tdpp} = s.Src_{tdpp}$.

   (e) $u.RecSim.zval = s.Rec.zval$.

   (f) $u.Src_{zval}.chosenval = s.Rec.zval$.

   (g) $u.Adv = s.Adv$.

   (h) $u.Env = s.Env$.

2. For every $u \in supp(lstate(\epsilon_2))$:
   If $u.RecSim.zval = \perp$ then one of the following holds:

   (a) For every $s \in supp(lstate(\epsilon_1))$, $s.Src_{yval}.chosenval = \perp$.

   (b) For every $s \in supp(lstate(\epsilon_1))$, $s.Rec.yval = \perp$, and $lstate(\epsilon_1).Src_{yval}.chosenval$ is the uniform distribution on $(\{0,1\} \rightarrow D)$.

   (c) $lstate(\epsilon_1).Rec.yval$ is the uniform distribution on $(\{0,1\} \rightarrow D)$.

11.3 The mapping proof

Lemma 11.4 *The relation R defined in Section 11.2 is a simulation relation from $RS\|Env$ to $SIS\|Env$. Furthermore, for each step of $RS\|Env$, the step correspondence yields at most two steps of $SIS\|Env$, that is, for every S, T, $|corrtasks(S,T)| \leq 2$.*

Proof. We prove that R satisfies the two conditions in Lemma 3.55. The start condition is shown as in the previous proofs. For the step condition, we define $corrtasks(RS_{RS\|Env} \times RA_{RS\|Env}) \rightarrow RA^*_{SIS\|Env}$ as follows:
For any $(S,T) \in (RS_{RS\|Env} \times RA_{RS\|Env})$:

- If T is any task of $RS\|Env$ except for $\{choose-rand_{yval}\}$, $\{rand_{yval}\}$, or $\{fix-zval_{Rec}\}$, then $corrtasks(S,T) = T$.

- If $T \in \{\{choose-rand_{yval}\}, \{rand_{yval}\}$, then $corrtasks(S,T) = \lambda$.

- If $T = \{fix-zval_{Rec}\}$ then $corrtasks(S,T) = \{choose-rand_{zval}\}\ \{rand_{zval}\}$.

Thus, each task of $RS\|Env$ that is locally-controlled by a common component ($Trans$, Adv, Src_{tdpp}, or Env) is replicated in $SIS\|Env$. For the locally-controlled tasks of Rec, there are three cases: $\{send(2,*)_{Rec}\}$, $\{fix-zval_{Rec}\}$, and $\{out(*)_{Rec}\}$. We map $\{send(2,*)_{Rec}\}$ to the same task of $RecSim$, $\{fix-zval_{Rec}\}$ to the two tasks of Src_{zval}, $choose-rand_{zval}$ followed by $rand(*)_{zval}$, and $\{out(*)_{Rec}\}$ to the same task of $Funct$. Finally, we map the locally-controlled tasks of Src_{yval} to λ.

All parts of the correspondence: enabling, preservation of Property 1, state equivalence, and trace distribution equivalence, are straightforward. $\square$

Proof. (Of Lemma 11.2:)
By Lemma 11.4, R is a simulation relation from $RS_k \| Env$ to $SIS_k \| Env$. Then Theorem 3.53 implies that $tdists(RS_k \| Env) \subseteq tdists(SIS_k \| Env)$. Since Env was chosen arbitrarily, this implies (by definition of $\leq_0$) that $RS_k \leq_0 SIS_k$. □

Proof. (Of Lemma 11.3:)
By Lemma 11.4, R is a simulation relation from $RS_k \| Env$ to $SIS_k \| Env$ for which $|corrtasks(S,T)| \leq 2$ for every S and T. Since that lemma holds for every k and every Env, Theorem 3.86 implies that $\overline{RS} \leq_{neg,pt} \overline{SIS}$. □

11.4 Putting the pieces together

Proof. (of Theorem 11.1):
Lemma 11.3 implies that $\overline{RS} \leq_{neg,pt} \overline{SIS}$. Since the simulator $SSim_k$ satisfies the constraints for a simulator in Figure 2, this implies that $\overline{RS} \leq_{neg,pt} \overline{IS}$. □

12 Correctness Proof, Case 4: Both Parties Corrupted

Theorem 12.1 *Let $\overline{RS}$ be a real-system family for $(\overline{D}, \overline{Tdp}, C)$, $C = \{Trans, Rec\}$, in which the family $\overline{Adv}$ of adversary automata is polynomial-time-bounded.*
Then there exists an ideal-system family $\overline{IS}$ for $C = \{Trans, Rec\}$, in which the family $\overline{Sim}$ is polynomial-time-bounded, and such that $\overline{RS} \leq_{neg,pt} \overline{IS}$.

In this case, the simulator knows everything, and so it can just play the protocol naturally, without interacting with $Funct$. This proof does not need any intermediate levels.

12.1 Simulator structure

For each k, we define a structured simulator $SSim_k$ to be the same as the system RS. Thus, the components are:

1. $Trans(D_k, Tdp_k)$.
2. $Rec(D_k, Tdp_k, \{Trans, Rec\})$, with $out'(x)_{Rec}$ renamed to $out''(x)_{Rec}$.
3. $Src(\{0,1\} \rightarrow D_k)_{yval}$.
4. $Src(Tdpp_k)_{tdpp}$.
5. $Adv(D_k, Tdp_k, \{Trans, Rec\})$.

Env provides $in(x)_{Trans}$ to $Funct$, $Trans$, and Adv, and $in(i)_{Rec}$ outputs to $Funct$, Rec, and Adv. Env receives $out(x)_{Rec}$ outputs from Adv, which are copies of $out''(x)_{Rec}$ outputs from Rec to Adv. The outputs of Src_{tdpp} go both to $Trans$ and to Adv, and the outputs of Src_{yval} go both to Rec and to Adv.

Lemma 12.2 *For every k, $RS_k \leq_0 SIS_k$.*

Lemma 12.3 $\overline{Int2} \leq_{neg,pt} \overline{SIS}$.

In the rest of this subsection, we fix Env, an environment for RS_k and SIS_k. And we suppress mention of k.

12.2 State correspondence

Here we define the correspondence R from states of $RS \| Env$ to states of $SIS \| Env$, which we will show to be a simulation relation in Section 12.3. The state correspondence is essentially the identity. More accurately, we don't care about the state of *Funct*, but we require the identity mapping for the states of all the other components of *SIS*.

Let ϵ_1 and ϵ_2 be discrete probability measures on finite execution fragments of $RS \| Env$ and $SIS \| Env$, respectively, satisfying the following properties:

1. **Trace distribution equivalence:** $tdist(\epsilon_1) = tdist(\epsilon_2)$.
2. **State equivalence:** There exists state equivalence classes $S_1 \in RS_{RS \| Env}$ and $S_2 \in RS_{Int1 \| Env}$ such that $supp(lstate(\epsilon_1)) \subseteq S_1$ and $supp(lstate(\epsilon_2)) \subseteq S_2$.

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) $u.Trans = s.Trans$.
   (b) $u.Rec = s.Rec$.
   (c) $u.Src_{tdpp} = s.Src_{tdpp}$.
   (d) $u.Src_{yval} = s.Src_{yval}$.
   (e) $u.Adv = s.Adv$.
   (f) $u.Env = s.Env$.

12.3 The mapping proof

Lemma 12.4 *The relation R defined in Section 12.2 is a simulation relation from $RS \| Env$ to $SIS \| Env$. Furthermore, for each step of $RS \| Env$, the step correspondence yields at most one step of $SIS \| Env$, that is, for every S, T, $|corrtasks(S,T)| \leq 1$.*

Proof. We prove that R satisfies the two conditions in Lemma 3.55. The start condition is shown as in the previous proofs. For the step condition, we define $corrtasks(RS_{RS \| Env} \times RA_{RS \| Env}) \rightarrow RA^*_{SIS \| Env}$ as follows:
For any $(S,T) \in (RS_{RS \| Env} \times RA_{RS \| Env})$:

- If T is any task of $RS \| Env$ except for $\{out'(x)_{Rec}\}$, then $corrtasks(S,T) = T$.
- If $T = \{out'(x)_{Rec}\}$, then $corrtasks(S,T) = \{out''(x)_{Rec}\}$.

Thus, the step correspondence is essentially the identity. Note that none of the *corrtasks* sequences includes any output or internal tasks of *Funct*; thus, *Funct* does not perform any locally-controlled steps in any of the executions that are obtained from the simulation relation.

All parts of the correspondence: enabling, preservation of Property 1, state equivalence, and trace distribution equivalence, are immediate.

□

Proof. (Of Lemma 12.2:)
By Lemma 12.4, R is a simulation relation from $RS_k \| Env$ to $SIS_k \| Env$. Then Theorem 3.53 implies that $tdists(RS_k \| Env) \subseteq tdists(SIS_k \| Env)$. Since *Env* was chosen arbitrarily, this implies (by definition of $\leq_0$) that $RS_k \leq_0 SIS_k$. □

Proof. (Of Lemma 12.3:)
By Lemma 12.4, R is a simulation relation from $RS_k \| Env$ to $SIS_k \| Env$ for which $|corrtasks(S,T)| \leq 1$ for every S and T. Since that lemma holds for every k and every *Env*, Theorem 3.86 implies that $\overline{RS} \leq_{neg,pt} \overline{SIS}$. □

12.4 Putting the pieces together

Proof. (of Theorem 12.1):
Lemma 12.3 implies that $\overline{RS} \leq_{neg,pt} \overline{SIS}$. Since the simulator $SSim_k$ satisfies the constraints for a simulator in Figure 2, this implies that $\overline{RS} \leq_{neg,pt} \overline{IS}$. □

13 Conclusions

Summary. In this paper, we have provided a complete model and correctness proof for a simple Oblivious Transfer protocol [GMW87], using Probabilistic I/O Automata (PIOAs) [SL95]. This involved modeling the protocol as a system of interacting PIOAs, and the properties that the protocol is intended to satisfy as another such system, and proving a formal correspondence between these two system models. We have considered four cases, based on which parties (transmitter and/or receiver) are corrupted. In all cases we have considered, the adversary is essentially an eavesdropper, not an active malicious participant.

The algorithm uses cryptographic primitives—specifically, a trap-door permutation and a hard-core predicate for its inverse. We have modeled the computational properties of these primitives in terms of PIOAs. The properties we have considered include both correctness of the output produced at the receiver end of the protocol, and secrecy of inputs and random choices of non-corrupted parties.

Following the usual proof methods for distributed algorithms, we have decomposed our proofs into several stages, with general transitivity results used to combine the results of the stages. 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 each proof.

Producing this proof required us to develop two new kinds of theory: First, we extended traditional PIOA theory in two ways:

- We defined a new notion of *tasks*, which provide a mechanism to resolve nondeterministic choices.
- We defined a new kind of *simulation relation*, which corresponds probability distributions on states at two levels of abstraction, and which allows splitting of distributions in order to show that individual steps preserve the correspondence.

Second, we developed a new theory for time-bounded PIOAs, specifically:

- We defined *time-bounded PIOAs*, which impose time bounds on the individual steps of the PIOAs.
- We defined a new *approximate, time-bounded, implementation relationship* between time-bounded PIOAs, which is sufficient to capture the typical relationships between cryptographic primitives and the abstractions they are supposed to implement.

In the multi-stage proofs, most of the stages represent exact (not approximate) implementations; we prove all these using standard PIOA theory, extended with our new simulation relation. The techniques for showing this are fairly standard in the distributed algorithms research literature, based on proving invariants and simulations by induction on the number of steps in an execution. The remaining stages involve replacement of a cryptographic primitive with a random counterpart; we prove that these satisfy our approximate implementation relationship. The techniques for showing this are based on recasting the definitions of the cryptographic primitives in terms of approximate implementation relationships, and then combining these primitives with other components in various ways that preserve the implementation relationships. Transitivity results allow us to combine all the implementation relationships proved at all the stages to obtain an overall approximate implementation relationship between the Oblivious Transfer algorithm and its property specification.

Evaluation. We believe that these methods provide a usable, scalable structure for carrying out complete, rigorous proofs of security protocols, assuming standard definitions for the cryptographic primitives that they use. The example illustrates how such proofs can be carefully broken down into manageable pieces, each piece proving a particular collection of facts. Various pieces use very different kinds of reasoning. Thus, typical "Distinguisher" arguments about cryptographic primitives (expressed in terms of implementation relationships) are isolated to certain stages of the proofs, whereas other stages use inductive, assertional methods.

Traditional formal reasoning about security protocols combines with this work as follows: We can model a system in which we use only abstract specifications for crypto primitives—for example, a system that uses OT as a building block. We can prove correctness of that system relative to the OT specification, using our simulation relation methods, or other methods such as model-checking. Then, we can "plug in" an OT implementation that implements the specification approximately (according to our approximate, time-bounded implementation relationship). Our general results about how this relationship is preserved with respect to composition imply that the resulting system approximately implements the system that has already been proved correct.

Future work. In this paper, the task scheduler is limited to be oblivious. It would be interesting to allow the task scheduler more power, by allowing it to be a function from some aspects of the previous history to the next scheduled task. The oblivious scheduler can be formulated equivalently in this way, where the available history information is just the sequence of past tasks. However, we would like to allow the scheduler more information, for instance, the actual states of adversarial components (like *Adv*) in between all the tasks. Making an extension of this kind will require rather deep changes throughout our work, all the way bace to the basic theory of task-PIOAs, in Section 3.

We plan to test the power of these techniques by applying them to more security protocols, including protocols that use different cryptographic primitives, and protocols that have more powerful adversaries (active rather than passive; adaptive). A good example is a simple key exchange protocol that uses a basic signature scheme, and that is intended to work against an active adversary. We would also like to consider Oblivious Transfer protocols in the presence of more powerful adversaries.

We will explore reasoning about more complicated protocols, which involve composition of many sub-protocols (e.g., multiple instances of Oblivious Transfer, or a combination of key distribution and secret communication); the idea is to try to use our techniques on the pieces and combine them using our general composition results.

Some interesting security protocols do not use any cryptographic primitives, for example, protocols that achieve perfect zero-knowledge [GMR89]. For these, our basic PIOA techniques should work, without any need for reasoning about approximate implementations. We will consider basic zero-knowledge protocols, for example, for graph isomorphism.

We would like to use the general methods presented here to model other cryptographic primitives, and to capture the ways in which they can be combined and used in protocols. This will involve restating the definitions of those primitives in terms of approximate implementation relationships with respect to more abstract PIOAs. Expressing these primitives in this way should enable reformulating traditional Distinguisher arguments (which proceed by contradiction) as (positive) arguments about approximate implementation. After reformulating these primitives, it remains to analyze protocols that use the primitives, using our mapping techniques.

A Component Interaction Diagrams

The figures that appear in this section show how the system components are connected in each of the four cases we consider. The arrows that have no labels represent the arbitrary actions of the environment *Env*. The action names $send(m)_{Trans}$, $receive(m)_{Trans}$, $send(m)_{Rec}$ and $receive(m)_{Rec}$ are abbreviated to, respectively, $s(m)_{Trans}$, $r(m)_{Trans}$, $s(m)_{Rec}$ and $r(m)_{Rec}$. In these figures, we abbreviate subscript *Trans* by just T and subscript *Rec* by just R.

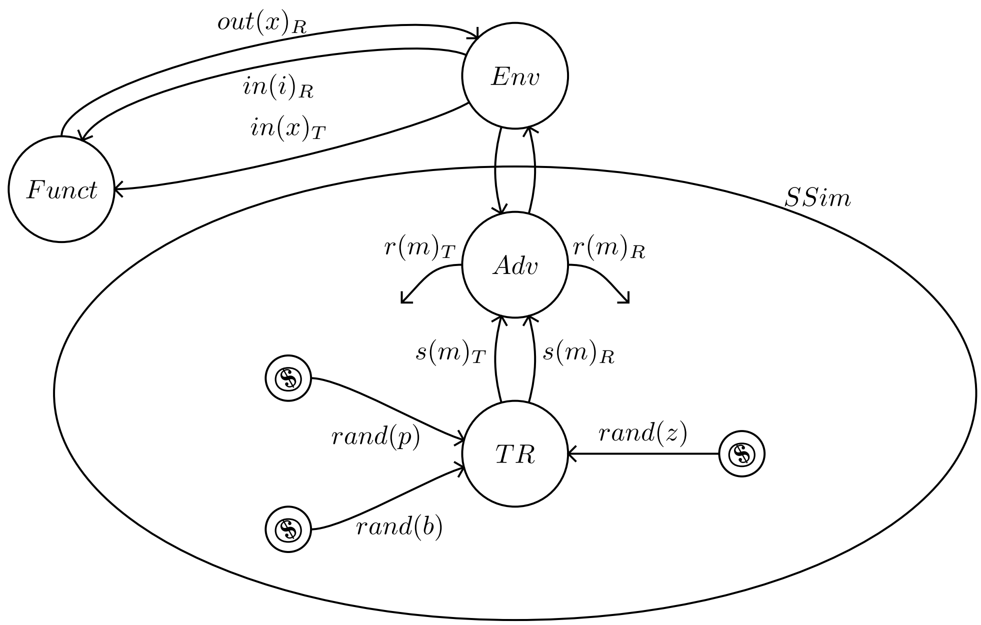

Figure 20: $SIS(\emptyset)$

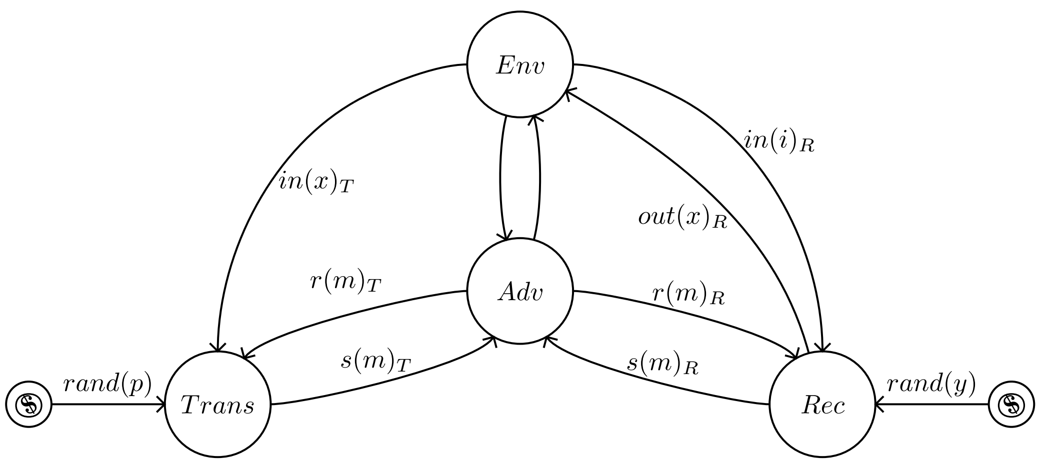

Figure 21: $RS(\emptyset)$

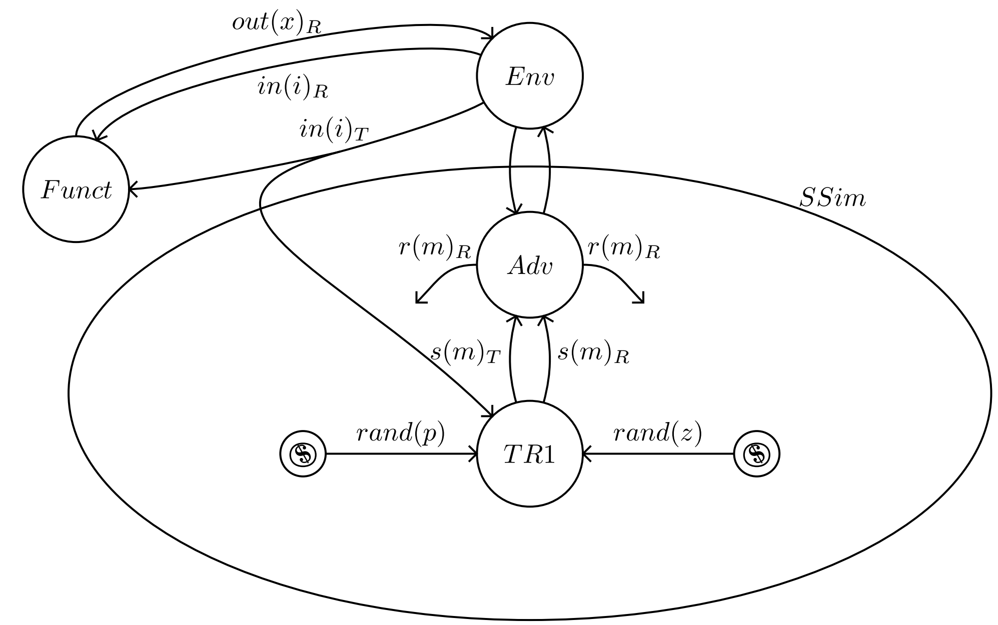

Figure 22: $Int1$ where neither party is corrupted

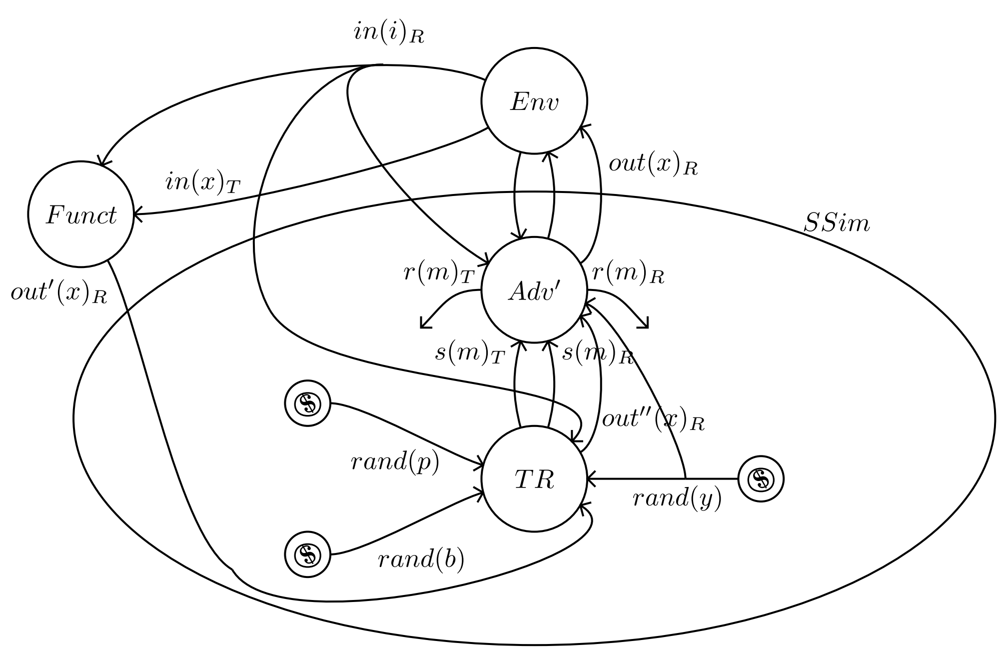

Figure 23: $SIS(\{Rec\})$

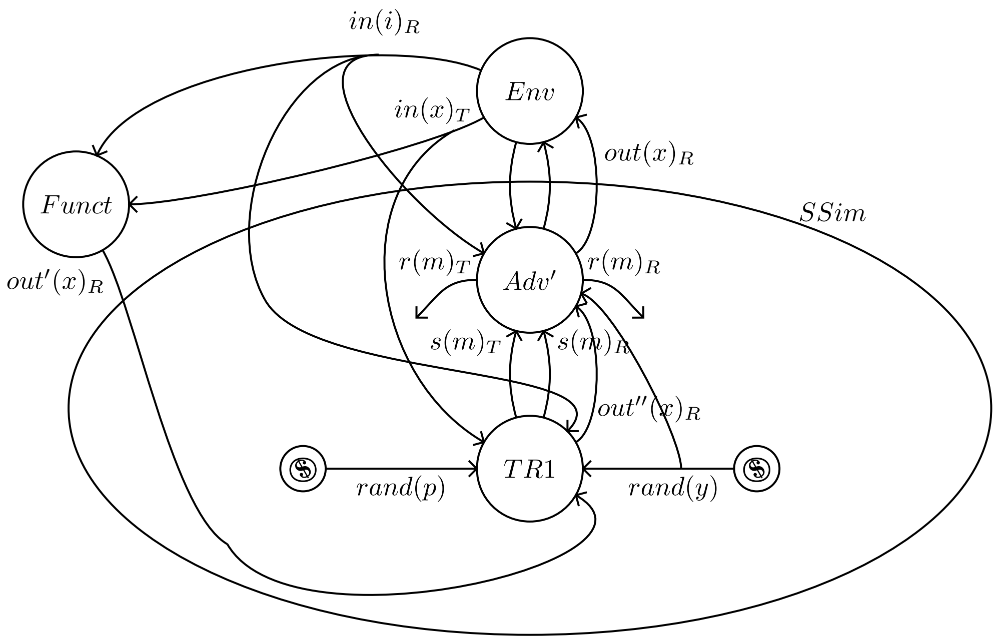

Figure 24: $Int1$ where only the Receiver is corrupted

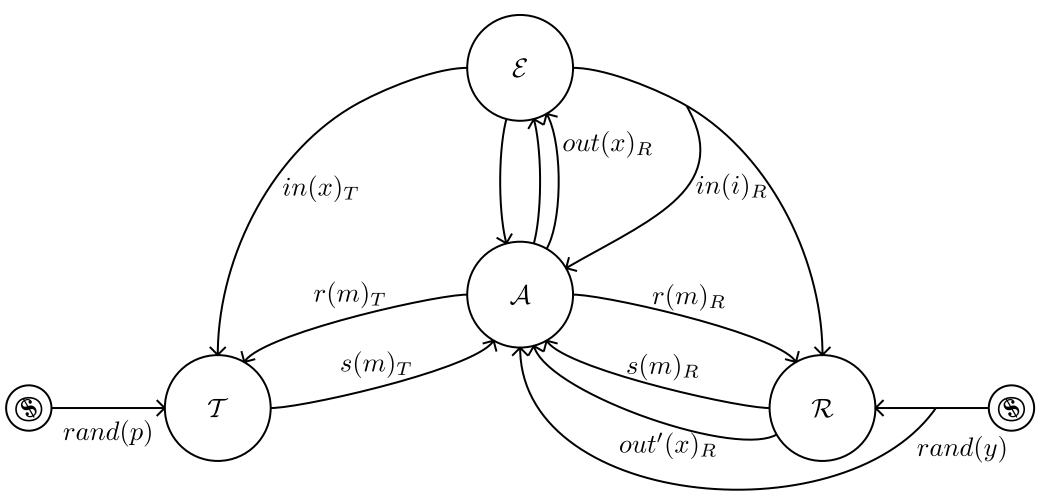

Figure 25: $RS(\{Rec\})$

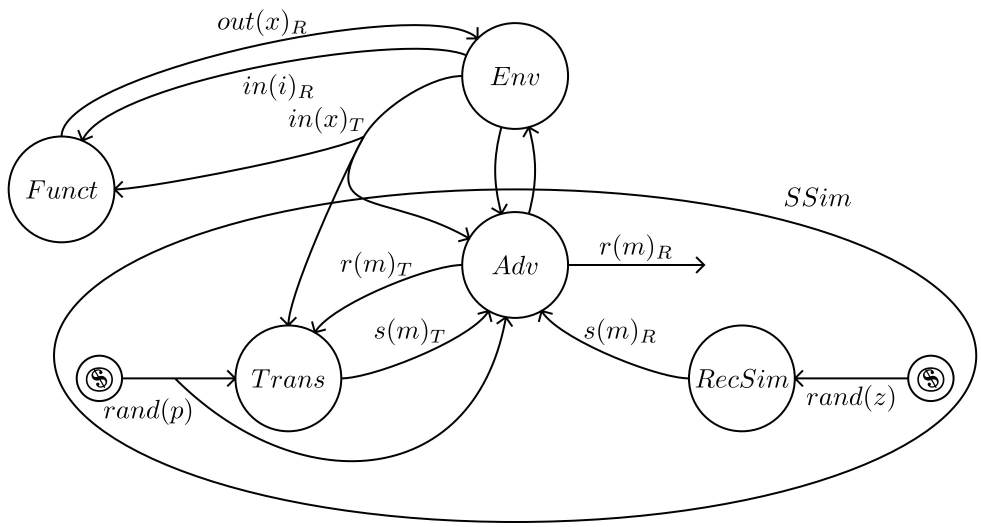

Figure 26: $SIS(\{Trans\})$

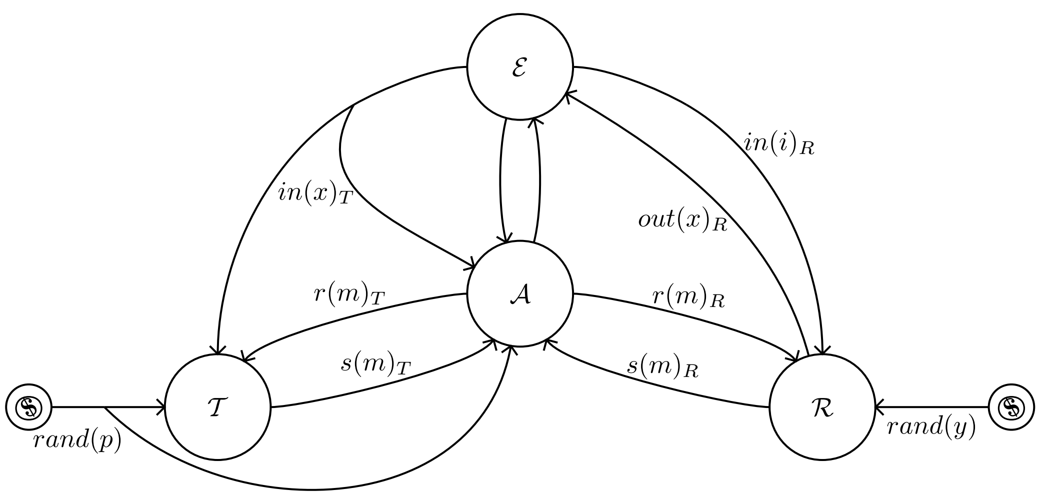

Figure 27: $RS(\{Trans\})$

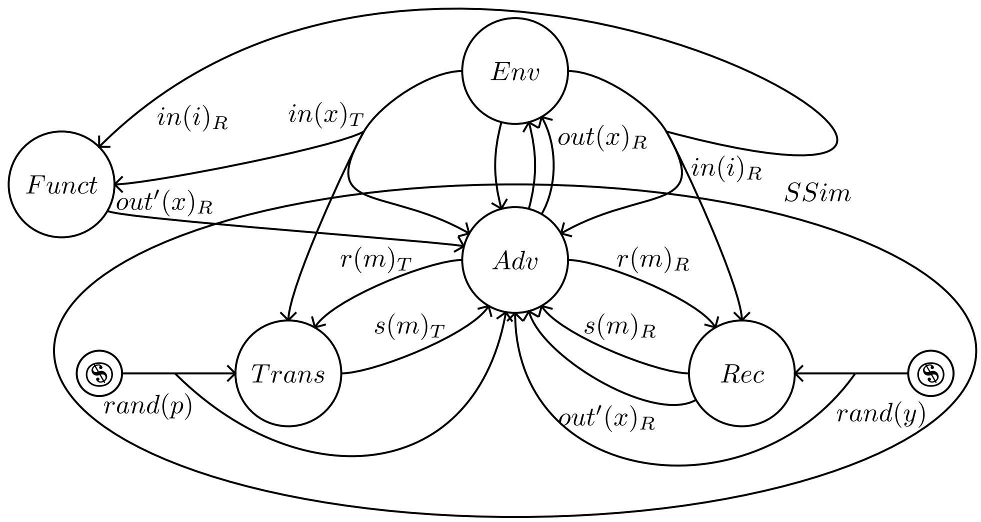

Figure 28: $SIS(\{Trans, Rec\})$

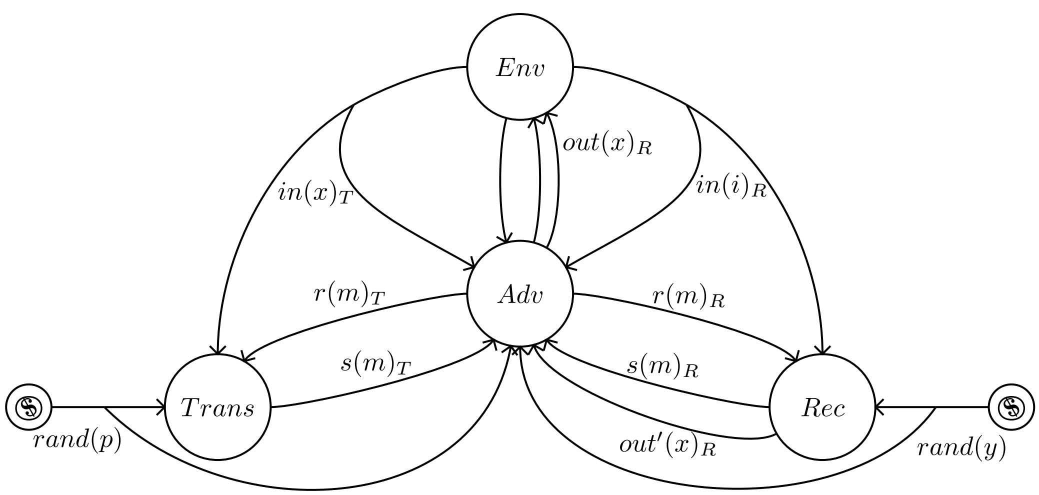

Figure 29: $RS(\{Trans, Rec\})$

Acknowledgments: We thank Frits Vaandrager for very useful preliminary discussions about how to model the Oblivious Transfer protocol and its requirements. We also thank Michael Ben-Or, Susan Hohenberger, and Ron Rivest for information, perspective, and advice regarding computational cryptography.

Ran Canetti is supported by NSF CyberTrust Grant 0430450. Ling Cheung is supported by DFG/NWO bilateral cooperation project Validation of Stochastic Systems (VOSS2). Dilsun Kaynar and Nancy Lynch are supported by NSF Award CCR-0326277 and DARPA/AFOSR MURI Award F49620-02-1-0325. Olivier Pereira is a Postdoctoral Researcher of the FNRS. Part of this work was done when he was a visiting scientist at MIT. Roberto Segala is supported by MIUR Project AIDA.

References

[AR00] M. Abadi and P. Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). *J. Cryptology* 15(2): 103-127 (2002). Preliminary version at *International Conference on Theoretical Computer Science IFIP TCS 2000,* LNCS, 2000.

[BPW03] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *10th ACM conference on computer and communications security (CCS)*, 2003. Extended version at the eprint archive, http://eprint.iacr.org/2003/015/.

[B05] B. Blanchet. A Computationally Sound Automatic Prover for Cryptographic Protocols. Presentation at the workshop on the link between formal and computational models. Paris, France. June 2005. http://www.di.ens.fr/~blanchet/talks/WorkshopLFCM.pdf.

[C01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Available at http://eprint.iacr.org/2000/067. Extended abstract in *42nd FOCS*, 2001.

[CH04] R. Canetti and J. Herzog. Universally Composable Symbolic Analysis of Cryptographic Protocols (The case of encryption-based mutual authentication and key exchange). Eprint archive, http://eprint.iacr.org/2004/334.

[DY83] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[EGL85] S. Even, O. Goldreich and A. Lempel, A randomized protocol for signing contracts, *CACM,* vol. 28, No. 6, 1985, pp. 637-647.

[GOLDREICH03] Oded Goldreich. *Foundations of Cryptography.* Volume I Basic Tools, Cambridge University Press, 2001, reprint of 2003, p. 64.

[GL89] O. Goldreich, L. A. Levin. A Hard-Core Predicate for all One-Way Functions. *21st Symposium on Theory of Computing (STOC),* ACM, 1989, pp. 25-32.

[GMW87] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game. *19th Symposium on Theory of Computing (STOC),* ACM, 1987, pp. 218-229.

[GMR89] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Comput.,* Vol. 18, No. 1, 1989, pp. 186-208.

[H05] S. Halevi. A plausible approach to computer-aided cryptographic proofs. eprint.iacr.org/2005/181. 2005.

[HMS03] D. Hofheinz and J. Mueller-Quade and R. Steinwandt. Initiator-Resilient Universally Composable Key Exchange. *ESORICS,* 2003. Extended version at the eprint archive, eprint.iacr.org/2003/063.

[JL91] B. Jonsson and K.G. Larsen. Specification and Refinement of Probabilistic Processes. *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, pages 266-277, Amsterdam, July 1991.

[K89] J. Kilian Uses of Randomness in Algorithms and Protocols, Chapter 3, The ACM Distinghished Dissertation 1989, MIT press.

[LSV03] Nancy Lynch, Roberto Segala and Frits Vaandrager. Compositionality for Probabilistic Automata. *CONCUR 2003: Concurrency Theory (The 14th International Conference on Concurrency Theory*, Marseille, France, September 2003). LNCS vol. 2761, pages 208-221, 2003. 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.

[MW04] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In the *1st Theory of Cryptography Conference (TCC)*, LNCS 2951, pp. 133–151. 2004.

[PW00] B. Pfitzmann, M. Waidner. Composition and Integrity Preservation of Secure Reactive Systems. In the *7th ACM Conference on Computer and Communications Security*, ACM 2000, 245-254.

[R81] M. Rabin. How to exchange secrets by oblivious transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.

[Segala95] Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* Ph.D Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, May 1995. Also, MIT/LCS/TR-676.

[SL95] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing,* Vol. 2. No. 2, pp 250-273, 1995.

[S02] V. Shoup. OAEP Reconsidered. *J. Cryptology*, 15(4): 223-249. 2002.

[S04] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. eprint.iacr.org/2004/332. 2004.