# Analyzing Security Protocols Using Time-Bounded Task-PIOAs*

Ran Canetti[1,2], Ling Cheung[2], Dilsun Kaynar[3], Moses Liskov[4], Nancy Lynch[2], Olivier Pereira[5], and Roberto Segala[6]

[1] IBM T.J. Watson Center
[2] Massachusetts Institute of Technology
[3] Carnegie-Mellon University
[4] The College of William and Mary
[5] Université catholique de Louvain
[6] Università di Verona

Version of February 16, 2007

**Abstract.** This paper presents the Time-Bounded Task-PIOA modeling framework, an extension of the Probabilistic Input/Output Automata (PIOA) framework that can be used for modeling and verifying security protocols. Time-bounded task-PIOAs can describe probabilistic and nondeterministic behavior, as well as time-bounded computation. Together, these features support modeling of important aspects of security protocols, including secrecy requirements and limitations on the computational power of adversarial parties. They also support security protocol verification using methods that are compatible with less formal approaches used in the computational cryptography research community. We illustrate the use of our framework by outlining a proof of functional correctness and security properties for a well-known Oblivious Transfer protocol.

## 1 Introduction

Modeling frameworks for interacting abstract state machines, such as I/O Automata, have long been used successfully for proving correctness of distributed algorithms, using proof techniques based on invariant assertions, levels of abstraction and composition. Security protocols are special cases of distributed

* Canetti's work on this project was supported by NSF CyberTrust Grant #0430450. Cheung was supported by DFG/NWO bilateral cooperation project 600.050.011.01 Validation of Stochastic Systems (VOSS) and by NSF Award #CCR-0326227. Kaynar and Lynch were supported by DARPA/AFOSR MURI Award #F49620-02-1-0325, MURI AFOSR Award #SA2796PO 1-0000243658, NSF Awards #CCR-0326277 and #CCR-0121277, and USAF, AFRL Award #FA9550-04-1-0121, and Kaynar was supported by US Army Research Office grant #DAAD19-01-1-0485. Pereira was supported by the Belgian National Fund for Scientific Research (F.R.S.-FNRS), and Segala by MIUR project AIDA and by INRIA project ProNoBiS.

algorithms—ones that use cryptographic primitives such as encryption and signature, and that guarantee properties such as secrecy and authentication. Thus, one should expect that the same kinds of models and techniques will be useful for proving properties of security protocols. However, making this approach work requires additions to the traditional frameworks, including mechanisms for modeling secrecy requirements, and for describing limitations on the knowledge and computational power of adversarial parties.

In this paper, we describe a modeling framework, the *Time-Bounded Task-PIOA* framework, that extends Segala's Probabilistic I/O Automata (PIOA) framework [Seg95,SL95,LSV] and supports description of security-related features. Time-bounded task-PIOAs directly model probabilistic and nondeterministic behavior, partial-information scheduling, and time-bounded computation. We define an *approximate implementation relation* for time-bounded task-PIOAs, $\leq_{neg,pt}$, which captures the notion of *computational indistinguishability*—the idea that a polynomial-time-bounded observer cannot (except with negligible probability) distinguish the behavior of one automaton from that of another. We show that $\leq_{neg,pt}$ is transitive and compositional. We also define a type of *probabilistic simulation relation*—a kind of step-by-step correspondence between task-PIOAs—that can be used to prove $\leq_{neg,pt}$. We believe these features will be useful for formal modeling and verification of many security protocols, using methods that are compatible with the informal approaches used in the computational cryptography research community.

We illustrate the use of our framework by outlining the correctness proof of a well-known two-party Oblivious Transfer (OT) protocol [EGL85,GMW87]. Our modeling involves two systems of automata.

- The *real system* consists of two automata representing the *protocol parties* and one automaton representing an *adversarial communication service*. This adversary has access to all messages sent during execution of the protocol.
- The *ideal system* consists of an ideal *Oblivious Transfer functionality* automaton, which specifies the allowed input/output behavior for Oblivious Transfer, and a *simulator* automaton, which interacts with the functionality and tries to mimic the behavior of the real system (e.g., messages between the protocol parties).

Correctness of this OT protocol is formulated as the statement that the real system implements the ideal system in the sense of $\leq_{neg,pt}$. That is, every possible behavior of the real protocol can be simulated by the abstract system containing the ideal functionality for OT. This property represents the standard cryptographic requirement that for any real-life adversary, there exists an ideal adversary (a simulator) such that the behavior of the system consisting of the protocol together with the real-life adversary is indistingushable by an external observer from the system consisting of the ideal functionality and the ideal adversary. In particular, the above property guarantees *functional correctness*: the input/output behavior of the protocol conforms to what is specified by the OT functionality. It also guarantees *security*, since access to protocol messages does

not give the real adversary any significant new knowledge or power (compared to the simulator which has no such access).

In order to prove that the Oblivious Transfer protocol implements the ideal system, we define a series of intermediate system models, and prove that each consecutive pair of models satisfies $\leq_{neg,pt}$. (Here the transitivity of $\leq_{neg,pt}$ is used.) This approach decomposes the security proof into several stages, each of which addresses some particular aspects of the protocol. In particular, reasoning about cryptographic primitives and limitations in computational power is isolated to a single stage in the proof, where a system using difficult-to-compute *hard-core bits* of trap-door functions is shown to implement a system using random bits. For this interesting step, we reformulate the standard notion of hard-core bits in terms of $\leq_{neg,pt}$, and prove that our reformulation is equivalent to the standard definition. The proof for this stage then uses this reformulated definition of hard-core bits plus composition results for $\leq_{neg,pt}$. Other stages are proved using probabilistic simulation relations.

**Background:** Security protocol verification is an active research area. Traditionally, security protocols have been verified using one of two approaches, often called *formal* and *computational*. In the *formal approach*, cryptographic operations are modeled symbolically, and security of a protocol is expressed in terms of absolute guarantees when the protocol is run against a Dolev-Yao adversary [DY83], which is incapable of breaking the cryptographic primitives. This approach lends itself to rigorous proofs using familiar methods; however, it neglects important computational issues that could render protocols invalid in practice. In contrast, in the *computational approach*, cryptographic operations are modeled as algorithms operating on bit strings, and security is expressed in terms of probabilistic guarantees when protocols are run against resource-bounded adversaries. This approach treats computational issues realistically, but it does not easily support rigorous proofs. For example, resource-bounded protocol components are commonly modeled as (probabilistic) Interactive Turing Machines (ITMs) [GMR89,Can01], but rigorous proofs in terms of ITMs are infeasible, because ITMs provide only a very low-level mechanism for representing computer programs. Another research direction, represented by [AR02,MW04,CH06] for instance, connects the formal to the computational approach, by proving general theorems asserting that, under certain circumstances, proofs carried out using the formal approach can be modified systematically to work also in the computational setting.

A recent trend in security verification is to combine formal and computational analysis in one framework (e.g., [LMMS98,PW00,BPW03,BPW04b,BCT04,RMST04,Bla05]), by defining computational restrictions on abstract machines. Our work follows this general approach, though with its own unique set of modeling choices, as we explain throughout this paper.

Our mathematical starting point is Segala's *Probabilistic Input/Output Automata (PIOA)* modeling framework [Seg95,SL95,LSV], which was originally developed for analyzing probabilistic distributed algorithms (e.g., [PSL00,SV99]).

PIOAs are abstract machines that may perform both probabilistic and nondeterministic choices; nondeterministic choice is used to select the next transition, and probabilistic choice is used to determine the resulting state. Nondeterminism is an essential feature of any framework for analyzing distributed algorithms, for several reasons:

- Nondeterminism makes it possible to define algorithms in a *general form*, leaving inessential choices (such as the order of certain events) unspecified. A proof of functional correctness for a nondeterministic algorithm carries over automatically to any algorithm obtained by resolving the nondeterministic choices in any particular way.
- Nondeterminism is needed in automata that are used as *high-level specifications* of allowed system behavior. Such specifications should express only restrictions that are really necessary, with options expressed using nondeterministic choices.
- Nondeterminism makes it possible to *avoid mathematical clutter*, in the form of unnecessary restrictions, in algorithm descriptions, theorems, and proofs. The resulting simplicity makes the entire verification enterprise easier.
- Nondeterminism is *unavoidable* anyway in systems of asynchronously interacting components. Even if all individual system components are purely probabilistic, when they are combined into a larger system, nondeterminism arises because the order in which components perform their steps is unspecified.

However, in order to state and prove probabilistic properties for distributed algorithms, one needs a way of *resolving the nondeterministic choices*. Segala does this by combining a PIOA with a *perfect-information scheduler*, which can use full knowledge about the past execution in selecting the next transition. But this scheduling mechanism is too powerful to use with security protocols: for example, a scheduler's choice of the next transition may depend on "secret" information hidden in the states of honest protocol participants, and thus may reveal information about secrets to corrupted participants.

For this reason, we define a less powerful scheduling mechanism for PIOAs: actions are grouped into *tasks*, and scheduling is carried out using arbitrary *task schedule* sequences. The combination of a PIOA and a task classification is called a *Task-PIOA* [CCK$^+$06a,CCK$^+$06b]. We think of a task schedule as simply a way of representing the order in which different system activities happen to occur. We consider this order to be determined accidentally, for example, by variations in speeds of different system components, or by unpredictable network delays, rather than by a purposeful scheduler entity. At first, it may seem that completely nonadaptive task sequences are insufficient to describe adversarial scheduling patterns of the sort that occur in security protocols. However, we model such patterns in a different way, which we explain in Section 3.1.

Task-PIOAs support familiar methods of abstraction and composition. They also include an implementation relation, $\leq_0$, between automata, based on trace distributions, and probabilistic simulation relations that can be used to prove $\leq_0$ relationships.

Finally, we augment the task-PIOA framework with notions of time bounds, expressed in terms of bit-string encodings of automata constituents—states, actions, etc. This yields the *Time-Bounded Task-PIOA* framework, which allows us to define generic notions such as polynomial-time-bounded task-PIOAs and the approximate implementation relation $\leq_{neg,pt}$. Computational indistinguishability assumptions, which are crucial for stating and proving the correctness of many cryptographic protocols, can be expressed in terms of $\leq_{neg,pt}$. We will present an example of computational indistinguishability in Section 5.

Our task-PIOA framework, and our models and proofs for Oblivious Transfer, have evolved somewhat over the past two years. In our initial technical report [CCK+06c], we used a version of task-PIOAs that is considerably more restrictive than the one presented here, because it imposes more consistency conditions on the tasks. An effect of these restrictions is that the automata have little power to modify their behavior dynamically, based on what has occurred so far in an execution. In fact, these restrictions meant that we were unable to model certain types of conditionally-branching adversaries that occur in security protocols, as was pointed out to us by Silvio Micali. This led us to generalize the framework to its current form, which appears in this paper and also in [CCK+06a,CCK+06b,CCK+05].

On the other hand, the original technical report [CCK+06c] completely analyzed four separate cases for the Oblivious Transfer algorithm, based on which protocol parties are assumed to be corrupted. Our newer version of the OT proof, in [CCK+05], analyzes only the most interesting of the four cases—the case where the Receiver protocol party is corrupted.

**Comparison with some related work:** Our formulation of computational security and our analysis of Oblivious Transfer follow the general style of results by Canetti on *Universally Composable (UC) Security* [Can01] and by Pfitzmann and Waidner on *Universal Reactive Simulatability (RSIM)* [PW01]. These, in turn, evolved from less-formal presentations in the computational cryptography community [GMW87,GMR89].

The RSIM work imposes computational restrictions on abstract machines for the purpose of analyzing security protocols [PW00,PW01,BPW04b]. RSIM abstract machines are interrupt-driven state machines that interact via a system of ports and buffers, and computational restrictions are expressed by relating these machines to probabilistic polynomial time (PPT) Interactive Turing Machines. A fixed distributed protocol, in which machines activate each other by generating explicit *clock* signals, is used for scheduling among the machines. This mechanism is similar to the ones typically used for ITMs [GMR89,Can01].

The models in [PW00,PW01,BPW04b] exhibit little or no nondeterminism: individual machines are purely probabilistic up to occurrences of inputs. Therefore, any *closed* collection of machines (i.e., with no further inputs) generates a unique probabilistic run, provided the machines are activated according to the scheduling algorithm mentioned above. In comparison, our framework makes extensive use of nondeterminism, allowing individual machines to make

nondeterministic choices *in a way that remains unknown to the adversary.* We then quantify over arbitrary (though non-adaptive) task schedules. As discussed in [CCLP07], this new treatment of the underlying concurrency leads to a notion of security that is incomparable to existing ones. Moreover, the use of traditional proof methods such as invariants and simulation relations are more prominent in our approach. For instance, we define a new type of simulation relation, tailored for task-based scheduling, and we give formal proofs that our simulation relations are sound for proving implementation relationships.

Another similar line of work is the *Probabilistic Polynomial-time Process Calculus (PPC)* of Mitchell et al. [LMMS98,MMS03,RMST04]. In this work, a process-algebraic language is used to specify protocols and their components. The terms of the algebra are restricted so that they can represent only probabilistic polynomial time (PPT) protocols. Security properties are specified using an asymptotic observational equivalence on process terms, which captures the notion of computational indistinguishability of the functions represented by these terms.

The PPC work follows the style of traditional concurrency theory: for example, the abstract machines underlying this process language are composable, and support equivalence proofs based on probabilistic bisimulation relations. Also, this work does allow nondeterministic choices, both within and among components. These nondeterministic choices are resolved by probabilistic schedulers of several different kinds, including special Markov chains and probability distributions on the set of enabled actions. Various restrictions, such as environment-independence and history-independence, are imposed on these schedulers in order to support computational security arguments.

Again, a main difference between our framework and PPC lies in the scheduling semantics. Schedulers for PPC are typically state-dependent, compared with our oblivious task schedules. Moreover, PPC schedulers do not resolve nondeterministic choices involving internal transitions. This is because the operational semantics for PPC is usually defined in such a way that internal computations are prioritized over external communications. If nondeterministic choices between internal transitions appear, they are taken with equal probability. As a result, when two PPC processes are related using an implementation relation, the same *external* scheduler can be used. In our case, task schedules also specify the ordering of internal events, which means that different task schedules must be used in order to match two different processes.

Overall, our work differs from RSIM and PPC in our particular choices of underlying machine model and scheduling mechanism, as well as the description of computational restrictions. We differ also in our emphasis on a particular modeling and proof methodology, derived from the one typically used for distributed algorithms: we use nondeterminism extensively as a means of abstraction, decompose our system descriptions using composition and levels of abstraction, and carry out proofs using invariants and simulation relations.

In other related work, security analysis is sometimes carried out, often informally, in terms of a sequence of *games* [Sho04,BR04,Bla06,Hal05]. These games are similar to our levels of abstraction (cf. Section 7).

**Overview of the paper:** Sections 2 and 3 review the PIOA and Task-PIOA frameworks, respectively, illustrating them with examples taken from the Oblivious Transfer case study. Section 4 defines time-bounded task-PIOAs, and the approximate implementation relation $\leq_{neg,pt}$. Section 5 illustrates our treatment of computational indistinguishability assumptions, by presenting our definition, based on $\leq_{neg,pt}$, of hard-core predicates for trap-door functions. Section 6 explains how we model cryptographic protocols and their requirements, illustrating this method with the OT protocol. Section 7 outlines our proofs for OT. Conclusions follow in Section 8.

Full definitions, results, and proofs for basic PIOAs and task-PIOAs appear in [CCK+06b], which is the full version of [CCK+06a]. Complete details for time-bounded task-PIOAs, $\leq_{neg,pt}$, hard-core predicates, and the Oblivious Transfer case study appear in the report [CCK+05]. Our earlier version of task-PIOAs, with its full proof for OT, appears in [CCK+06c].

## 2 Probabilistic I/O Automata

In this section, we summarize basic definitions and results for PIOAs; full definitions, results, and proofs appear in [CCK+06a,CCK+06b]. We illustrate the definitions with examples extracted from the Oblivious Transfer case study.

### 2.1 Mathematical notation

We write $\mathsf{N}$ for the set of natural numbers and $\mathsf{R}^{\geq 0}$ for the sets of nonnegative real numbers. If $X$ is any set, then we denote the set of finite sequences and infinite sequences of elements from $X$ by $X^*$ and $X^\omega$, respectively. If $\rho$ is a sequence then we use $|\rho|$ to denote the length of $\rho$. We use $\lambda$ to denote the empty sequence (over any set). If $\rho \in X^*$ and $\rho' \in X^* \cup X^\omega$, then we write $\rho \circ \rho'$ for the concatentation of the sequences $\rho$ and $\rho'$. Sometimes, when no confusion seems likely, we omit the $\circ$ symbol, writing just $\rho\rho'$.

### 2.2 Probability measures

In this section, we review basic definitions for probability measures. A $\sigma$-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 $\sigma$-field over $X$, is called a *measurable space*. A measure on a measurable space $(X, \mathcal{F})$ is a function $\mu : \mathcal{F} \to [0, \infty]$ that is countably additive: for each countable family $\{X_i\}_i$ of pairwise disjoint elements of $\mathcal{F}$, $\mu(\cup_i X_i) = \sum_i \mu(X_i)$. A *probability measure* on $(X, \mathcal{F})$ is a measure on $(X, \mathcal{F})$ such that $\mu(X) = 1$.

A *discrete probability measure* on a set $X$ is a probability measure $\mu$ on $(X, 2^X)$, such that, for each $C \subseteq X$, $\mu(C) = \sum_{c \in C} \mu(\{c\})$. We define $Disc(X)$ to be, the set of discrete probability measures on $X$. In the sequel, we often omit the set notation when we denote the measure of a singleton set. For a discrete probability measure $\mu$ on a set $X$, $supp(\mu)$ denotes the support of $\mu$, that is, the set of elements $x \in X$ such that $\mu(x) \neq 0$. Given set $X$ and element $x \in X$, the *Dirac* measure $\delta(x)$ is the discrete probability measure on $X$ that assigns probability 1 to $x$.

If $\{\rho_i\}_{i \in I}$ is 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 $\rho$ 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 $\mu_1, \mu_2$ 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 $\mu$ 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 $\mu$ under $f$.

### 2.3 Operations involving probability measures:

Now we define three operations involving probability measures: *flattening*, *lifting*, and *expansion*. We will use these in Section 3.5 to define a probabilistic simulation relation. These three operations have previously been defined, for example, in [LSV]. The first operation, which we call *flattening*, takes a discrete probability measure over probability measures and "flattens" it into a single probability measure.

**Definition 1.** *Let $\eta$ be a discrete probability measure on $Disc(X)$. Then the* flattening *of $\eta$, denoted by $flatten(\eta)$, is the discrete probability measure on $X$ defined by $flatten(\eta) = \sum_{\mu \in Disc(X)} \eta(\mu)\mu$.*

The second operation, which we call *lifting*, takes a relation $R$ between two domains $X$ and $Y$ and "lifts" it to a relation between discrete measures over $X$ and $Y$. Informally speaking, a measure $\mu_1$ on $X$ is related to a measure $\mu_2$ on $Y$ if $\mu_2$ can be obtained by "redistributing" the probabilities masses assigned by $\mu_1$, in such a way that relation $R$ is respected.

**Definition 2.** *The* lifting *of $R$, denoted by $\mathcal{L}(R)$, is the relation from $Disc(X)$ to $Disc(Y)$ defined by: $\mu_1$ $\mathcal{L}(R)$ $\mu_2$ iff there exists a* weighting function *$w : X \times Y \to \mathbf{R}^{\geq 0}$ such that the following hold:*

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

Finally, we define our third operation, called *expansion*. Expansion is defined in terms of flattening and lifting, and. is used directly in our new definition of simulation relations. The *expansion* operation takes a relation between discrete measures on two domains $X$ and $Y$, and returns a relation of the same kind that relates two measures whenever they can be decomposed into two $\mathcal{L}(R)$-related measures.

**Definition 3.** *Let $R$ be a relation from $Disc(X)$ to $Disc(Y)$. The* expansion *of $R$, denoted by $\mathcal{E}(R)$, is a relation from $Disc(X)$ to $Disc(Y)$. It is defined by: $\mu_1\ \mathcal{E}(R)\ \mu_2$ iff there exist two discrete measures $\eta_1$ and $\eta_2$ on $Disc(X)$ and $Disc(Y)$, respectively, such that the following hold:*

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

Informally speaking, we enlarge $R$ by adding pairs of measures that can be "decomposed" into weighted sums of measures, in such a way that the weights can be "redistributed" in an $R$-respecting manner. This is used in our definition of a simulation relation, in Section 3.5.

### 2.4 Probabilistic I/O Automata

Here we review the definitions of Probabilistic I/O Automata, their executions and traces, and their composition and hiding operations.

**Definition 4.** *A* Probabilistic I/O Automaton (PIOA) *$\mathcal{P}$ is a tuple $(Q, \bar{q}, I, O, H, D)$, where:*

- *$Q$ is a countable set of* states*, with* start state *$\bar{q} \in Q$.*
- *$I$, $O$ and $H$ are countable, pairwise disjoint sets of actions, referred to as* input, output and internal actions*, respectively. The set $A := I \cup O \cup H$ is called the* action alphabet *of $\mathcal{P}$. The set of* external *actions of $\mathcal{P}$ is $E := I \cup O$, and the set of* locally controlled *actions is $L := O \cup H$.*
- *$D \subseteq Q \times (I \cup O \cup H) \times Disc(Q)$ is a* transition relation*, where $Disc(Q)$ is the set of discrete probability measures on $Q$.*

*An action $a$ is* enabled *in a state $q$ if $(q, a, \mu) \in D$ for some $\mu$. If $I = \emptyset$, then $\mathcal{P}$ is* closed*. We assume that $\mathcal{P}$ satisfies the following properties:*

- ***Input enabling:** For every $q \in Q$ and $a \in I$, $a$ is enabled in $q$.*
- ***Transition determinism:** For every $q \in Q$ and $a \in A$, there is at most one $\mu \in Disc(Q)$ such that $(q, a, \mu) \in D$.*

**Definition 5.** *An* execution fragment *of $\mathcal{P}$ is a finite or infinite sequence $\alpha = q_0\ a_1\ q_1\ a_2\ \ldots$ of alternating states and actions, such that:*

1. *If $\alpha$ is finite, it ends with a state.*

2. *For every non-final $i$, there is a transition $(q_i, a_{i+1}, \mu) \in D$ with $q_{i+1} \in supp(\mu)$.*

*We write $fstate(\alpha)$ for $q_0$, and if $\alpha$ is finite, we write $lstate(\alpha)$ for its last state. We use $\mathsf{Frags}(\mathcal{P})$ (resp., $\mathsf{Frags}^*(\mathcal{P})$) to denote the set of all (resp., all finite) execution fragments of $\mathcal{P}$. An* execution *of $\mathcal{P}$ is an execution fragment $\alpha$ with $fstate(\alpha) = \bar{q}$. $\mathsf{Execs}(\mathcal{P})$ (resp., $\mathsf{Execs}^*(\mathcal{P})$) denotes the set of all (resp., all finite) executions of $\mathcal{P}$. The* trace *of an execution fragment $\alpha$, written $\mathsf{trace}(\alpha)$, is the restriction of $\alpha$ to the external actions of $\mathcal{P}$. We say that $\beta$ is a* trace *of $\mathcal{P}$ if there is $\alpha \in \mathsf{Execs}(\mathcal{P})$ with $\mathsf{trace}(\alpha) = \beta$.*

A PIOA, together with a *scheduler* that chooses the sequence of actions to be performed, gives rise to a unique *probabilistic execution*, and thereby, to a unique probability distribution on traces. Traditionally, the schedulers used for PIOAs have been perfect-information schedulers, which can use full knowledge about the past execution in selecting the next transition. Next, we define composition of PIOAs:

**Definition 6.** *Two PIOAs $\mathcal{P}_i = (Q_i, \bar{q}_i, I_i, O_i, H_i, D_i)$, $i \in \{1, 2\}$, are said to be* compatible *if $A_i \cap H_j = O_i \cap O_j = \emptyset$ whenever $i \neq j$. That is, the two automata do not share any output actions, and no internal action of either is an action of the other. In that case, we define their* composition *$\mathcal{P}_1 \| \mathcal{P}_2$ to be the PIOA $(Q_1 \times Q_2,\ (\bar{q}_1, \bar{q}_2),\ (I_1 \cup I_2) \setminus (O_1 \cup O_2),\ O_1 \cup O_2,\ H_1 \cup H_2, D)$, where $D$ is the set of triples $((q_1, q_2), a, \mu_1 \times \mu_2)$ such that:*

1. *$a$ is enabled in some $q_i$.*
2. *For $i \in \{1, 2\}$, if $a \in A_i$ then $(q_i, a, \mu_i) \in D_i$, and otherwise $\mu_i = \delta(q_i)$.*

Our definition of composition can be generalized to any finite number of PIOAs. Note that if an input of one PIOA is an output of another, then it becomes an output action of the composed automaton. A *hiding* operator is also available for PIOAs:

**Definition 7.** *Given $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$ and $S \subseteq O$, $hide(\mathcal{P}, S)$ is defined to be $(Q, \bar{q}, I, O \setminus S, H \cup S, D)$.*

### 2.5 Examples

In this subsection, we give four examples of PIOAs, all derived from our Oblivious Transfer (OT) case study.

**Random sources:** Our first example is a *random source* PIOA $Src(D, \mu)$, which simply chooses and outputs a single value, obtained from a given probability measure $\mu$ over a given domain $D$. We use such random sources to encapsulate random choices made by the two parties in our OT protocol model.

$Src(D, \mu)$ has no input actions. It has one internal action, *chooserand*, by which it chooses a random number, and a set of output actions, $\{rand(d) | d \in$

**Automaton** $Src(D, \mu)$:
**Signature:**
Input:
none
Internal:
*chooserand*
Output:
$rand(d), d \in D$

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

**Transitions:**

*chooserand*
Precondition:
$chosenval = \perp$
Effect:
$chosenval :=$ choose-random$(D, \mu)$

$rand(d)$
Precondition:
$d = chosenval$
Effect:
none

**Fig. 1.** Code for random source PIOA $Src(D, \mu)$

$D\}$, by which it outputs a chosen value. We assume that $Src(D, \mu)$ performs *chooserand* exactly once, but may output the chosen value any number of times.

Figure 1 contains a specification of a *random source* PIOA $Src(D, \mu)$ using *precondition-effects* (guarded command) notation. We first present the PIOA's *signature*, that is, its list of actions, classified as input, output, or internal. We next describe the state, in terms of *state variables*; in this case, we have only one state variable, *chosenval*, representing the chosen random value. This variable is initialized to the special value $\perp$, which indicates that no value has yet been chosen. Finally, we present *transition definitions*, each of which describes the transitions associated with a particular kind of action. Internal action *chooserand* is enabled to occur in any state in which $chosenval = \perp$; its effect is to set *chosenval* to a value chosen randomly, according to measure $\mu$. Output action $rand(d)$ is enabled when $d = chosenval$; it has no effect on the state, which means that it may happen any number of times.[7]

The $Src(D, \mu)$ PIOA makes probabilistic choices, in the effects of its *chooserand* transitions. Nondeterminism appears in the form of uncertainty as to how many times the *rand* output is performed.

**Oblivious Transfer functionality:** Our next example is a PIOA *Funct*, which we use in Sections 6.1 and 6.4 to specify functional correctness and security requirements for Oblivious Transfer. The specification, and the protocol, distinguish two endpoints of the protocol, which we call the *Transmitter end* and the *Receiver end*; these correspond to the two parties that engage in the protocol.

[7] In [CCK+06c], the $rand(d)$ transition definition has an additional precondition, saying that $chosenval \neq \perp$. This is unnecessary, however, because the $rand(d)$ action itself is defined, in the signature, only if $d \in D$. We have eliminated other such redundant conditions elsewhere in this paper.

Code for *Funct* is provided in Figure 2. *Funct* has two kinds of inputs, one at the Transmitter end and one at the Receiver end. The Transmitter inputs are of the form $in(x)_{Trans}$, where $x$ is a pair of input bits; for technical convenience, we have written this pair as a *mapping* from the index set $\{0,1\}$ to the set of possible bit values $\{0,1\}$. Input $in(x)_{Trans}$ represents the arrival of the bit pair $x$ at the Transmitter endpoint, from the external environment. The Receiver inputs are of the form $in(i)_{Rec}$, where $i$ is simply a bit, in $\{0,1\}$.

*Funct* also has one kind of output, $out(w)_{Rec}$, which occurs at the Receiver endpoint, where parameter $w$ is simply a bit. This output $w$ should be one of the two Transmitter input bits, either $x(0)$ or $x(1)$, depending on the value of the Receiver input $i$: if inputs $in(x)_{Trans}$ and $in(i)_{Rec}$ occur, the output bit $w$ should be $x(i)$. In other words, functional correctness for OT means that the Receiver end of the protocol should output just one of the Transmitter's input bits—the one specified by the Receiver's input.

The state of *Funct* consists of values for two variables, $xval$ and $ival$, which simply keep track of the two kinds of inputs. Each kind of input should normally occur only once, so the PIOA ignores any subsequent input arrivals of either kind. As in the *Src* PIOA, outputs may occur any number of times. Output $out(w)_{Rec}$ is enabled if both the Transmitter input $x$ and the Receiver input $i$ have already occurred, and $w$ is the correct bit, calculated by applying the mapping $x$ to the index $i$: $w = xval(ival)$.

*Funct* makes no probabilistic choices. Nondeterminism appears in the order in which the inputs occur, and in how many times the various kinds of inputs and outputs occur. Our definition differs slightly from the one in [CLOS02], which imposes more sequential behavior: There, the Receiver input is considered only if the Transmitter input occurred first. Here, nondeterminism provides a natural way to express the notion that the two inputs may occur in either order, and the output is produced only after both inputs have been received.

**Oblivious Transfer protocol parties:** Finally, we present two PIOAs representing the Transmitter and Receiver parties of the OT protocol. They assume:

- $D$, a fixed domain of values, and
- $Tdpp$, a set of *trap-door permutation pairs* for $D$, and $Tdp$, the corresponding set of *trap-door permutations*.
- $B$, a *hard-core predicate* for $Tdpp$.

A *trap-door permutation* of $D$ is a permutation that is easy to compute but hard to invert without knowledge of special "trap-door" information (see [Gol01] for the standard definition). We use $Tdp$ to denote the set of trap-door permutations for $D$, and $Tdpp$ to denote the set of pairs $p = (f, f^{-1})$, where $f \in Tdp$; that is, $Tdpp$ is the set of pairs consisting of a trap-door permutation and its inverse. For $p \in Tdpp$, we sometimes write $p.funct$ and $p.inv$ to indicate $f$ and $f^{-1}$, respectively. A *hard-core predicate* $B$ for the trap-door permutation $f$ is a mapping from $D$ to $\{0,1\}$, which satisfies the following condition: when $B$ is applied to $f^{-1}(z)$, where $z$ is randomly chosen, the result is indistinguishable (except with negligible probability) from a uniformly chosen random value. Again,

*Funct* :

**Signature:**

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

Output:
- $out(w)_{Rec}$, $w \in \{0,1\}$

**State:**

- $xval \in (\{0,1\} \to \{0,1\}) \cup \{\perp\}$, initially $\perp$
- $ival \in \{0,1,\perp\}$, initially $\perp$

**Transitions:**

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

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

$out(w)_{Rec}$
Precondition:
- $xval, ival \neq \perp$
- $w = xval(ival)$

Effect:
- none

**Fig. 2.** Code for Oblivious Transfer functionality *Funct*

see [Gol01] for the standard definition. Also see Section 5 for a reformulation of this definition in our style, which is useful in our OT proof.

Informally speaking, the protocol works as follows. The Transmitter uses a random source to choose a random trap-door permutation pair $p = (f, f^{-1})$. Then, it sends a round 1 message to the Receiver, containing just the function $f$, but not the inverse $f^{-1}$. The Receiver chooses a random pair $y$ of elements of $D$, again using a random source. After it has chosen $y$, and has received its own input $i$ and the round 1 message, the Receiver computes a new pair $z$ of elements of $D$; this computation involves applying $f$ to the element of the pair with index $i$, but leaving the other element unchanged. Then, the Receiver sends a round 2 message to the Transmitter, containing $z$.

Once the Transmitter has received its own input $x$ and the round 2 message, it computes a pair $b$ of bits. In doing this, it treats the two indices of the pairs identically: for each $i \in \{0,1\}$, it computes $B(f^{-1}(z(i)))$ and exclusive-or's ($\oplus$) the result with $z(i)$. Then, it sends a round 3 message to the Receiver, containing the pair $b$. Finally, once the Receiver has received the round 3 message, it extracts the desired bit $w$ by a simple calculation, $wval := b(ival) \oplus B(yval(ival))$, and outputs $w$.

We model this protocol using two separate PIOAs, one for the Transmitter and one for the Receiver. The code for the Transmitter, $Trans(D, Tdp)$, appears in Figure 3. The input actions for *Trans* are the $x$ inputs from the external environment, the *rand* inputs from a random source of trap-door permutation pairs, and the *receive* actions for round 2 messages. The output actions are the *send* actions for round 1 and 3 messages. The only internal action is the $fix - bval$ action, by which the Transmitter computes the $b$ pair. The state

records the input pair (in $xval$), the trap-door permutation pair (in $pval$), and the $z$ and $b$ pairs (in $zval$ and $bval$, respectively).

An $in(x)_{Trans}$ transition simply records the input $x$ in $xval$, if it is the first time such an input has arrived. Likewise, a $rand(p)_{pval}$ transition records the trap-door permutation pair $p$, assumed to be arriving from a random source, in $pval$. A $send(1, f)$ transition sends a round 1 message containing $p.funct$; this may happen any number of times. A $receive(2, z)$ transition records a $z$ pair that arrives in a round 2 message, in $zval$. A $fix - bval$ transition computes the pair $b$ and stores the result in $bval$. This calculation is the same for both indices $i$, and involves applying $p.inv$ to $z(i)$, applying the hard-core predicate $B$, and xor'ing the result with the input $x(i)$. Finally, a $send(3, b)$ transition sends a round 3 message containing the calculated $b$.

The *Trans* PIOA makes no probabilistic choices on its own, though it receives the results of such choices from an external random source, by means of $rand(p)_{pval}$ actions. On the other hand, *Trans* does exhibit some nondeterminism: for example, it may send a round 1 message at any time after receiving $p$, before or after receiving the $x$ input. In contrast, in traditional presentations of such protocols, the inputs are usually assumed to arrive at the start, before the protocol computations and communications begin; however, this restriction is not essential. *Trans* may also send its messages any number of times.

The code for the Receiver, $Rec(D, Tdp)$ appears in Figure 4. The input actions are the $i$ inputs from the external environment, the $rand$ inputs from a random source of $D$ pairs, and the $receive$ actions for round 1 and 3 messages. The outputs are the $send$ actions for round 2 messages, and the final $out(w)$ outputs. The only internal action is the $fix - zval$ action, by which the Receiver computes the $z$ pair from the $y$ pair. The state records the input (in $ival$), the received trap-door function (in $fval$), the $y$ and $z$ pairs (in $yval$ and $zval$), and the output (in $wval$).

An $in(i)_{Rec}$ transition records the input $i$ in $ival$, and a $rand(y)_{yval}$ records the random input $y$ in $yval$. A $receive(1, f)$ transition records the trap-door permutation $f$ that arrives in a round 1 message, in $fval$. A $fix - zval$ transition computes the $z$ pair from the $y$ pair; this calculation involves applying $f$ to $y(i)$ but leaving $y(1 - i)$ unchanged. A $send(2, z)$ transition sends a round 2 message. A $receive(3, b)$ transition receive a $b$ value that arrives in a round 2 message, and uses it to calculate the output $w$ value, which is stored in $wval$. Finally, an $out(w)$ transition outputs the computed $w$ value. The Receiver PIOA makes no probabilistic choices, but does make nondeterministic choices with respect to how many times it sends its messages.

## 3 Task-PIOAs

PIOAs, as presented in Section 2, may exhibit scheduling nondeterminism. In order to state and prove probabilistic properties, we must resolve all such nondeterminism. However, the perfect-information schedulers that have previously been used to resolve nondeterministic choices in PIOAs are too powerful for com-

$Trans(D, Tdp)$:

**Signature:**

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

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

Internal:
- $fix-bval_{Trans}$

**State:**
- $xval \in (\{0,1\} \to \{0,1\}) \cup \{\bot\}$, initially $\bot$
- $pval \in Tdpp \cup \{\bot\}$, initially $\bot$
- $zval \in (\{0,1\} \to D) \cup \{\bot\}$, initially $\bot$
- $bval \in (\{0,1\} \to \{0,1\}) \cup \{\bot\}$, initially $\bot$

**Transitions:**

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

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

$send(1,f)_{Trans}$
Precondition:
- $pval \neq \bot$
- $f = pval.funct$

Effect:
- none

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

$fix-bval_{Trans}$
Precondition:
- $xval, pval, zval \neq \bot$
- $bval = \bot$

Effect:
- for $i \in \{0,1\}$ do
  - $bval(i) = B(pval.inv(zval(i))) \oplus xval(i)$

$send(3,b)_{Trans}$
Precondition:
- $b = bval$

Effect:
- none

**Fig. 3.** Code for the Transmitter, $Trans(D, Tdp)$

putational analysis of security protocols; for example, a scheduler's choice of the next action may depend on information hidden in the states of honest protocol participants, and thus may reveal information about the secrets to corrupted participants. To avoid this problem, we resolve nondeterminism using a more restrictive, non-adaptive *task schedule* mechanism.

In this section, we define task-PIOAs, which are simply PIOAs plus a classification of actions into *tasks*. We use tasks as units of scheduling: we define *task schedules*, and describe how a task schedule resolves nondeterministic choices, thereby generating a probabilistic execution. Also in this section, we define a simple *perfect implementation* relationship between task-PIOAs, and a type of probabilistic simulation relation for task-PIOAs that can be used to prove that one task-PIOA perfectly implements another. We illustrate our definitions with

$Rec(D, Tdp)$ :

**Signature:**

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

Output:
- $send(2,z)_{Rec}$, $z \in (\{0,1\} \to D)$
- $out(w)_{Rec}$, $w \in \{0,1\}$

Internal:
- $fix - zval_{Rec}$

**State:**
- $ival \in \{0,1,\perp\}$, initially $\perp$
- $fval \in Tdp \cup \{\perp\}$, initially $\perp$
- $yval, zval \in (\{0,1\} \to D) \cup \{\perp\}$, initially $\perp$
- $wval \in \{0,1,\perp\}$, initially $\perp$

**Transitions:**

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

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

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

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

Effect:
- $zval(ival) := fval(yval(ival))$
- $zval(1 - ival) := yval(1 - ival)$

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

Effect:
- none

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

$out(w)_{Rec}$
Precondition:
- $w = wval$

Effect:
- none

**Fig. 4.** Code for the Receiver, $Rec(D, Tdp)$

examples from the OT case study. Full definitions, results, and proofs for task-PIOAs appear in [CCK+06a,CCK+06b].

### 3.1 Basic definitions

**Definition 8.** *A* task-PIOA $\mathcal{T}$ *is a pair* $(\mathcal{P}, R)$*, where* $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$ *is a PIOA (satisfying the transition determinism and input enabling properties), and* $R$ *is an equivalence relation on the locally-controlled actions* $L = O \cup H$*. The equivalence classes of* $R$ *are called* tasks*, and we say that a task* $T$ *is* enabled *in state* $q$ *if some* $a \in T$ *is enabled in* $q$*. Unless otherwise stated, we will use terminology inherited from the PIOA setting.*

We require that every task-PIOA $\mathcal{T}$ satisfies the following axiom:

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

By virtue of this axiom and the transition determinism for PIOAs, tasks can be used to resolve all nondeterminism. That is, from a given state, specifying a task is sufficient to determine the next action, and therefore (by transition determinism), the next transition.

**Definition 9.** *A* task schedule *for* $\mathcal{T} = (\mathcal{P}, R)$ *is a finite or infinite sequence* $\rho = T_1\, T_2\, \ldots$ *of tasks in* $R$.

Thus, a task schedule is *non-adaptive*, in the sense that it does not depend on dynamic information generated during execution. Because of the action-determinism assumption for task-PIOAs and the transition-determinism assumption for PIOAs, $\rho$ can be used to generate a unique probabilistic execution, and hence, a unique trace distribution, of $\mathcal{P}$. One can do this by repeatedly scheduling tasks, each of which determines at most one transition of $\mathcal{P}$. This is captured formally as an "apply" operation: given a task sequence $T_1\, T_2\, \ldots$ and an execution fragment $\alpha$,

1. if $T_1$ is enabled in $lstate(\alpha)$, then, due to action- and transition-determinism, there is a unique transition from $lstate(\alpha)$ with an action label in $T$, and the result of "applying" $T_1$ to $\alpha$ is $\alpha$ extended with that unique transition;
2. if $T_1$ is not enabled in $lstate(\alpha)$, then the result of "applying" $T_1$ to $\alpha$ is $\alpha$ itself;
3. repeat with remaining $T_i$'s.

This construction can be generalized from a single execution fragment $\alpha$ to a discrete probability measure $\epsilon$ on execution fragments. See [CCK+06a,CCK+06b] for details.

In the special case where $\epsilon$ is the Dirac measure on the start state (i.e., $\delta(\bar{q})$), the result of applying any task schedule $\rho$ to $\delta(\bar{q})$ is said to be a *probabilistic execution* of $\mathcal{T}$. This can be viewed as a probabilistic tree generated by running $\mathcal{P}$ from its start state $\bar{q}$ and resolving nondeterministic choices according to $\rho$. The *trace distribution* induced by $\rho$, $tdist(\rho)$, is the image measure of $apply(\delta(\bar{q}), \rho)$ under the measurable function trace. A *trace distribution* of $\mathcal{T}$ is $tdist(\rho)$ for any $\rho$, and we define $tdists(\mathcal{T})$, the set of trace distributions of $\mathcal{T}$, to be $\{tdist(\rho) \mid \rho \text{ is a task schedule for } \mathcal{T}\}$.

Note that, although our task schedules are non-adaptive, they are *arbitrary*, rather than being determined by a fixed policy as in [Can01,PW01,BPW04b].

**Local schedulers:** In [CCK+06a,CCK+06b], we introduced a second mechanism for resolving nondeterminism, in addition to task schedules: *local schedulers*. Local schedulers resolve nondeterminism within system components, based on local information only. These allows us to remove the action determinism assumption, thus adding flexibility in automaton descriptions.

**Adversarial scheduling:** The standard scheduling mechanism in the security protocol community is an *adversarial scheduler*—a resource-bounded algorithmic entity that determines the next move adaptively, based on its own view of the computation so far. Our non-adaptive task schedules do not directly capture the adaptivity of adversarial schedulers. However, we model them in a different way, by separating scheduling concerns into two parts:

- We model an adaptive adversarial scheduler as a system component, such as a message delivery service that can eavesdrop on the communications and control the order of message delivery based on what it hears. Such a service has access to partial information about the execution: it sees information that other components communicate to it during execution, but not "secret information" that remain within these components' states. The adversary's choices may be essential to the analysis of the protocol.
- On the other hand, we resolve basic scheduling choices by a task schedule sequence, chosen nondeterministically in advance. These choices are less important; for example, in the OT protocol, both the Transmitter and Receiver make random choices, but it is inconsequential which does so first.

### 3.2 Composition and hiding operations

Task-PIOAs compose easily to yield another task-PIOA.

**Definition 10.** *Given compatible task-PIOAs* $\mathcal{T}_1 = (\mathcal{P}_1, R_1)$ *and* $\mathcal{T}_2 = (\mathcal{P}_2, R_2)$*, we define their* composition $\mathcal{T}_1 \| \mathcal{T}_2$ *to be the task-PIOA* $(\mathcal{P}_1 \| \mathcal{P}_2, R_1 \cup R_2)$*, that is, the composition of the underlying PIOAs combined with the union of the sets of tasks of the two component task-PIOAs.*

It is easy to check that action determinism is preserved under composition of PIOAs. Moreover, compatibility requires disjoint sets of locally controlled actions, therefore $R_1 \cup R_2$ is an equivalence relation. This guarantees the composition $\mathcal{T}_1 \| \mathcal{T}_2$ is a well-defined task-PIOA.

A composite task-PIOA may be scheduled using the same task schedule mechanism as for the component task-PIOAs. The difference is simply that the schedules of a composition include tasks of both component task-PIOAs, interleaved in arbitrary order. The hiding operation for task-PIOAs hides all specified output actions:

**Definition 11.** *Given task-PIOA* $\mathcal{T} = (\mathcal{P}, R)$ *and a set* $S \subseteq O$ *of output actions,* $hide(\mathcal{T}, S)$ *is defined to be* $(hide(\mathcal{P}, S), R)$*.*

### 3.3 Examples

When defining a task-PIOA based on a given PIOA, one may have options in defining the tasks, in particular, with respect to the *granularity* of the defined tasks. At one extreme, we might classify each action in a task by itself; at the other, we could make tasks as large as possible, subject to the action determinism requirement. We generally follow an intermediate strategy of grouping together actions that appear to represent the "same kind of activity". We illustrate this in the following examples.

**Random sources:** To convert the PIOA $Src(D, \mu)$ defined in Section 2.5 into a task-PIOA, we define two tasks: one consisting of the single internal action *chooserand*, and the other consisting of all the output actions, $\{rand(d) \mid d \in D\}$. Thus, a task schedule decides when to attempt to output a random element, but does not decide which element $d$ is to be output. The choice of which $d$ to output (if the *rand* task is enabled) is determined, not by the task, but by the value of the *chosenval* state variable. The task schedule does not need to "know" the value of $d$ in order to schedule the *rand* task. Moreover, since our task schedules are non-adaptive, the task schedule cannot choose when to attempt to output the random element based on the particular value of $d$ that resides in *chosenval*, or on any other dynamically-determined information. This means that our task schedules cannot leak dynamically-determined "secrets", such as the value of $d$, as traditional perfect-information schedulers can.

**Functionality:** To convert *Funct* defined in Section 2.5 into a task-PIOA, we group the output actions into a single task, $\{out(w)_{Rec} \mid w \in \{0,1\}\}$. The idea is that the task schedule should decide when to try to output, but should not decide which bit $w$ to output; moreover, its choice of when to try to output is not based on any dynamically-determined information such as the values of $x$, $i$, or $w$.

**Protocol parties:** For $Trans(D, Tdp)$, we define three tasks, corresponding to the three kinds of activities it performs, that is, sending the round 1 message, computing $b$, and sending the round 3 message:

- $\{send(1, f)_{Trans} \mid f \in Tdp\}$.
- $\{fix - bval_{Trans}\}$.
- $\{send(3, b)_{Trans} \mid b \in (\{0,1\} \to \{0,1\})\}$.

Likewise, for $Rec(D, Tdp)$, we define three tasks, for sending the round 2 message, computing $z$, and outputting $w$:

- $\{send(2, z)_{Rec} \mid z \in (\{0,1\} \to D)\}$.
- $\{fix - zval_{Rec}\}$.
- $\{out(w)_{Rec}\} \mid w \in \{0,1\}\}$.

### 3.4 Perfect implementation relation

An *implementation relation* between two automata expresses the idea that every behavior of one of the automata, in any environment, is also a behavior of the second, in the same environment. This notion makes sense only if the two automata interact with their environment via the same interface:

**Definition 12.** *Two task-PIOAs $\mathcal{T}_1$ and $\mathcal{T}_2$ are* comparable *if $I_1 = I_2$ and $O_1 = O_2$, that is, if they have the same input actions and the same output actions.*

We define the notion of an environment for a task-PIOA:

**Definition 13.** *If $\mathcal{T}$ and $\mathcal{E}$ are task-PIOAs, then $\mathcal{E}$ is said to be an* environment *for $\mathcal{T}$ if $\mathcal{T}$ and $\mathcal{E}$ are compatible and $\mathcal{T}\|\mathcal{E}$ is closed.*

Thus, an environment for $\mathcal{T}$ is an automaton that may be composed with $\mathcal{T}$ and that provides all of $\mathcal{T}$'s inputs.

**Definition 14.** *If $\mathcal{T}_1$ and $\mathcal{T}_2$ are comparable task-PIOAs, then we say that $\mathcal{T}_1$* perfectly implements *$\mathcal{T}_2$, written as $\mathcal{T}_1 \leq_0 \mathcal{T}_2$, provided that $tdists(\mathcal{T}_1\|\mathcal{E}) \subseteq tdists(\mathcal{T}_2\|\mathcal{E})$ for every task-PIOA $\mathcal{E}$ that is an environment for both $\mathcal{T}_1$ and $\mathcal{T}_2$. Equivalently, given any task schedule $\rho_1$ for $\mathcal{T}_1\|\mathcal{E}$, there is a task schedule $\rho_2$ for $\mathcal{T}_2\|\mathcal{E}$ such that $tdist(\rho_1) = tdist(\rho_2)$; that is, the two schedules yield the same trace distribution.*

The relation $\leq_0$ is called *perfect implementation* because it specifies exact equality of corresponding trace distributions. Later, in Sections 4.4 and 4.5, we will define approximate implementation relations.

Transitivity of $\leq_0$ is easy to see. A compositionality theorem for $\leq_0$ is proved in [CCK+06a,CCK+06b]:

**Theorem 1.** *Suppose that $\mathcal{T}_1$ and $\mathcal{T}_2$ are comparable task-PIOAs such that $\mathcal{T}_1 \leq_0 \mathcal{T}_2$, and $\mathcal{T}_3$ is a task-PIOA that is compatible with each of $\mathcal{T}_1$ and $\mathcal{T}_2$. Then $\mathcal{T}_1\|\mathcal{T}_3 \leq_0 \mathcal{T}_2\|\mathcal{T}_3$.*

We also have a theorem for hiding:

**Theorem 2.** *Suppose that $\mathcal{T}_1$ and $\mathcal{T}_2$ are comparable task-PIOAs such that $\mathcal{T}_1 \leq_0 \mathcal{T}_2$, Suppose that $S$ is a set of output actions of both $\mathcal{T}_1$ and $\mathcal{T}_2$. Then $hide(\mathcal{T}_1, \mathcal{S}) \leq_0 hide(\mathcal{T}_2, \mathcal{S})$.*

### 3.5 Simulation relations

*Simulation relations* provide sufficient conditions for proving that one automaton implements another, according to some precise notion of implementation. Typically, simulation relations reduce the proof obligations for implementation into conditions relating the start states and conditions relating individual steps. Checking these individual conditions is generally much easier, and more systematic, than reasoning about entire executions.

In [CCK+06a,CCK+06b], we defined a new type of probabilistic simulation relation for task-PIOAs, extending the ones presented by Segala for PIOAs in [Seg95,SL95,LSV]. We proved that the new simulation relations are sound for proving perfect implementation ($\leq_0$). Here, we provide an overview. Our definition uses the notion of *expansion* defined in Section 2.2.

We need two other auxiliary definitions. The first expresses consistency between a probability measure over finite executions and a task schedule: informally speaking, a measure $\epsilon$ over finite executions is said to be consistent with a task schedule $\rho$ if $\epsilon$ assigns non-zero probability only to those executions that are possible under the task schedule $\rho$. When matching the behaviors of two automata in a simulation relation, we use this condition to reduce proof obligations.

**Definition 15.** *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed task-PIOA, $\epsilon$ a discrete probability measure over finite executions of $\mathcal{P}$, and $\rho$ a finite task schedule for $\mathcal{T}$. Then we say that $\epsilon$ is* consistent with *$\rho$ provided that $supp(\epsilon) \subseteq supp(apply(\delta(\bar{q}), \rho))$.*

For the second definition, suppose we have a mapping $c$ that, given a finite task schedule $\rho$ and a task $T$ of a task-PIOA $\mathcal{T}_1$, yields a task schedule of another task-PIOA $\mathcal{T}_2$. The idea is that $c(\rho, T)$ describes how $\mathcal{T}_2$ matches task $T$, in situations where it has already matched the task schedule $\rho$. Using $c$, we define a new function $full(c)$ that, given a task schedule $\rho$, iterates $c$ on all the elements of $\rho$, thus producing a "full" task schedule of $\mathcal{T}_2$ that matches all of $\rho$.

**Definition 16.** *Let $\mathcal{T}_1 = (\mathcal{P}_1, R_1)$ and $\mathcal{T}_2 = (\mathcal{P}_2, R_2)$ be two task-PIOAs, and let $c : ({R_1}^* \times R_1) \rightarrow {R_2}^*$ be a function that assigns a finite task schedule of $\mathcal{T}_2$ to each finite task schedule of $\mathcal{T}_1$ and task of $\mathcal{T}_1$. Define $full(c) : {R_1}^* \rightarrow {R_2}^*$ recursively as follows: $full(c)(\lambda) := \lambda$, and $full(c)(\rho T) := (full(c)(\rho)) \circ (c(\rho, T))$ (that is, the concatenation of $full(c)(\rho)$ and $c(\rho, T)$).*

We can now define our probabilistic simulation relations for task-PIOAs. Note that our simulation relations are relations on probability measures on executions, rather than relations on states. We relate measures because of certain cases that arise in our OT proof, namely, cases where related random choices are made at different points during execution in the two related automata. See Section 3.6 for an example. Moreover, since we use oblivious task-based scheduling (instead of perfect-information scheduling), a typical implementation proof involves the construction of matching task schedules that preserve trace distributions. Therefore, it is more natural to reason with measures on executions, rather than measures on states.

**Definition 17.** *Let $\mathcal{T}_1 = (\mathcal{P}_1, R_1)$ and $\mathcal{T}_2 = (\mathcal{P}_2, R_2)$ be two comparable closed task-PIOAs. Let $R$ be a relation from $Disc(\mathsf{Execs}^*(\mathcal{P}_1))$ to $Disc(\mathsf{Execs}^*(\mathcal{P}_2))$, such that, if $\epsilon_1\ R\ \epsilon_2$, then $tdist(\epsilon_1) = tdist(\epsilon_2)$. Then $R$ is a simulation from $\mathcal{T}_1$ to $\mathcal{T}_2$ if there exists $c : ({R_1}^* \times R_1) \rightarrow {R_2}^*$ such that the following properties hold:*

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

Thus, the relationship between $\mathcal{T}_1$ and $\mathcal{T}_2$ is formulated using a relation $R$ between measures $\epsilon_1$ and $\epsilon_2$ on executions of the two automata. We require a priori that $R$ relate $\epsilon_1$ and $\epsilon_2$ only if they "look the same" when viewed externally, that is, if they yields the same probability measure on traces. The rest of the definition asserts a simple correspondence between initial states, and a step condition that is based on a correspondence rule $c$ for task schedules. The following theorem says that, for closed task-PIOAs, the existence of a simulation relation implies inclusion of sets of trace distributions. The main soundness result for (not-necessarily-closed) task-PIOAs then follows as a corollary, showing

that the existence of simulation relations for all environments implies perfect implementation.

**Theorem 3.** *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two comparable closed task-PIOAs. If there is a simulation relation $R$ from $\mathcal{T}_1$ to $\mathcal{T}_2$, then $tdists(\mathcal{T}_1) \subseteq tdists(\mathcal{T}_2)$.*

**Corollary 1.** *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two comparable task-PIOAs. Suppose that, for every environment $\mathcal{E}$ for both $\mathcal{T}_1$ and $\mathcal{T}_2$, there is a simulation relation $R$ from $\mathcal{T}_1 \mid \mathcal{E}$ to $\mathcal{T}_2 \mid \mathcal{E}$. Then $\mathcal{T}_1 \leq_0 \mathcal{T}_2$.*

Finally, the following corollary captures a special case of the simulation relation Definition 17. Any relation that satisfies the hypotheses of Corollary 2 is guaranteed to be a simulation relation. This is used directly in proving the correctness of the OT protocol.

**Corollary 2.** *Let $\mathcal{T}_1 = (\mathcal{P}_1, R_1)$ and $\mathcal{T}_2 = (\mathcal{P}_2, R_2)$ be two comparable closed task-PIOAs. Let $R$ be a relation from $Disc(\mathsf{Execs}^*(\mathcal{P}_1))$ to $Disc(\mathsf{Execs}^*(\mathcal{P}_2))$, satisfying the condition: if $\epsilon_1\ R\ \epsilon_2$ then $tdist(\epsilon_1) = tdist(\epsilon_2)$. Let $c : ({R_1}^* \times R_1) \rightarrow {R_2}^*$. Suppose further that the following conditions hold:*

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

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

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

### 3.6 Oblivious Transfer example

We refer the reader to [CCK+06a,CCK+06b] for a small example, which is derived from our first version of the OT case study [CCK+06c], in the case where only the Transmitter is corrupted. (It does not appear in the more recent version [CCK+05], since that includes only the case where the Receiver is corrupted.) This example illustrates the use of a simulation relation that corresponds probability measures on executions of two task-PIOAs, rather than just individual executions.[8]

Briefly, the example consists of two closed task-PIOAs, *Trap-door* and *Rand*. *Rand* performs two steps, first choosing a number $z$ uniformly at random from a fixed finite set of integers, and then outputting it; thus, it is essentially a

[8] The TR version [CCK+06b] repairs a small bug in the mapping definition in [CCK+06a].

special case of our random source automaton. *Trap-door* performs three steps, first choosing a random number $y$, then applying a known permutation $f$ to $y$, obtaining $z$, and finally outputting $z$.

We use a simulation relation $R$ to show that every trace distribution of *Trap-door* is also a trace distribution of *Rand*. In defining $R$, it appears most natural to correspond the steps that define $z$ in both automata; that is, the internal step of *Trap-door* that computes $z$ should correspond to the internal step of *Rand* that chooses $z$ randomly. This means that the internal step of *Trap-door* that chooses $y$ randomly should correspond to the empty sequence of steps of *Rand*. To express this action correspondence in terms of simulation relations, we had to correspond the point between the random choice of $y$ and the computation of $z$ in *Trap-door* to the initial state of *Rand*. Essentially, a probability distribution describing possible values of $y$, in *Trap-door*, is related to a single state of *Rand*. Our new simulation relation notion is flexible enough to handle this type of correspondence; the earlier notions, in [Seg95,SL95,LSV], were not.

## 4 Time-Bounded Task-PIOAs

A key assumption in computational cryptography is that certain problems cannot be solved with non-negligible probability by entities with bounded computational resources. This assumption is used to prove that protocols are secure against resource-bounded adversarial parties. Thus, any framework that can express such proofs needs mechanisms for expressing computational restrictions. As defined so far, task-PIOAs do not have any such mechanisms; in this section, we describe additions to task-PIOAs in order to describe computational resource bounds.

We consider two different kinds of restrictions: (1) *static bounds* on the size of the representation of a task-PIOA, and on what it can do in a single step, and (2) *dynamic bounds* on the number of steps that the task-PIOA performs. In nearly all work on computational cryptograhy, these static and dynamic bounds are combined into a single overall bound. For instance, Interactive Turing Machines representing protocol components are typically assumed to be *polynomial-time-bounded*, which means that the total amount of work they do, during the entire protocol lifetime, is bounded by a polynomial. Since each individual ITM step is assume to have tiny granularity, accessing a small number of tape squares and making small local changes, the significant restriction here is on the total number of steps an ITM performs. However, task-PIOAs are more abstract than ITMs, and may perform steps with larger granularity. This means that it is necessary to bound the amount of work that can be done in each step, as well as the number of steps. We believe it is meaningful to consider these two bounds separately, since they express different sorts of limitations. For example, in modeling *long-lived* security protocols [CM97,MQU07], it seems clear that limitations on what a machine can do in one step, or in a bounded amount of time, are quite different from limitations on the total lifetime of the machine.

To capture (1), the restrictions on representation and individual steps, we define the notion of a *time-bounded task-PIOA*. Such a machine has bounded-length bit-string representations of all of its constituents, and imposes time bounds on Turing machines that manipulate these representations, for example, decoding them and computing the next action and next state. To capture (2), the restrictions on the number of steps, we simply consider bounded-length schedules.

Basic bound definitions appear in Section 4.1, followed by extensions in Section 4.2 to capture generic notions such as "polynomial time". In Sections 4.4 and 4.5, we present new approximate implementation notions that take the bounds into account. Finally, our definition of simulation relations is revised, so that it can be used to prove approximate simulation. We omit proofs—the details appear in [CCK+05].

### 4.1 Basic definitions

Now we define time-bounded task-PIOAs. We assume a standard, globally known bit-string representation for actions and tasks of task-PIOAs.

**Definition 18.** *A task-PIOA $\mathcal{T}$ is said to be* $p$-time-bounded*, where $p \in \mathsf{R}^{\geq 0}$, provided:*

1. Automaton parts: *Every state and transition has a bit-string representation, and the length of the representation of every automaton part is at most $p$.*
2. Decoding: *There is a deterministic Turing machine that decides whether a given representation of a candidate automaton part is indeed such an automaton part, and this machine runs in time at most $p$. And similarly for deciding whether an action is in a task, or whether two actions are in the same task.*
3. Determining the next action: *There is a deterministic Turing machine that, given a state and a task of $\mathcal{T}$, determines the next action (or indicates "no action"), in time at most $p$.*
4. Determining the next state: *There is a probabilistic Turing machine that, given a state and an action of $\mathcal{T}$, determines the next state of $\mathcal{T}$, in time at most $p$.*

*Furthermore, each of these Turing machines can be described using a bit string of length at most $p$, according to some standard encoding of Turing machines.*

Composing two compatible time-bounded task-PIOAs yields another time-bounded task-PIOA with a bound that is linear in the sum of the original bounds.

**Lemma 1.** *There exists a universal constant $c_{comp}$ such that the following holds. Suppose $\mathcal{T}_1$ is a $p_1$-time-bounded task-PIOA and $\mathcal{T}_2$ is a $p_2$-time-bounded task-PIOA, where $p_1, p_2 \geq 1$. Then $\mathcal{T}_1 \| \mathcal{T}_2$ is a $c_{comp}(p_1+p_2)$-time-bounded task-PIOA.*

**Proof.** By a detailed analysis of Turing machine algorithms to carry out the various computations. See [CCK+05]. □

Similarly, hiding changes the bound by a linear in the time needed to recognize the hidden actions:

**Definition 19.** *Suppose $S$ is a set of actions of a time-bounded task-PIOA and $p \in \mathsf{R}^{\geq 0}$. We say that $S$ is $b$-time recognizable if there is a probabilistic Turing machine $M$ that, given the representation of a candidate action $a$, decides if $a \in S$. Furthermore, the machine $M$ runs in time less than $p$ and can be described by less than $p$ bits, according to some standard encoding of Turing machines.*

**Lemma 2.** *There exists a universal constant $c_{hide}$ such that the following holds. Suppose $\mathcal{T}$ is a $p$-time-bounded task-PIOA, where $p \in \mathsf{R}^{\geq 0}$, $p \geq 1$. Let $S$ be a $p'$-time recognizable subset of the set of output actions of $\mathcal{T}$. Then hide$(\mathcal{T}, S)$ is a $c_{hide}(p + p')$-time-bounded task-PIOA.*

We fix the constants $c_{comp}$ and $c_{hide}$ from now on in the paper. Finally, we define our bound for the number of steps:

**Definition 20.** *Let $\rho$ be a task schedule of task-PIOA $\mathcal{T}$ and let $q \in \mathsf{N}$ be given. Then $\rho$ is* $q$-bounded *if $|\rho| \leq q$, that is, if $\rho$ consists of at most $q$ tasks.*

### 4.2 Task-PIOA families

Security protocols are often parameterized by a *security parameter* $k \in \mathsf{N}$, which represents, for example, the length of a key used for encryption and decryption. For such a parameterized protocol, typical security claims say that the success probability of any resource-bounded adversarial entity in breaking the protocol diminishes quickly as $k$ increases. To express such claims, we define families of task-PIOAs indexed by a security parameter:

**Definition 21.** *A* task-PIOA family $\overline{\mathcal{T}}$ *is an indexed set $\{\mathcal{T}_k\}_{k \in \mathsf{N}}$ of task-PIOAs.*

Time bounds can also be expressed in terms of the security parameter.

**Definition 22.** *Given a function $p : \mathsf{N} \to \mathsf{R}^{\geq 0}$, we say that task-PIOA family $\overline{\mathcal{T}}$ is* $p$-time-bounded *if every $\mathcal{T}_k$ is $p(k)$ time-bounded. $\overline{\mathcal{T}}$ is* polynomial-time-bounded *provided that $\overline{\mathcal{T}}$ is $p$-time-bounded for some polynomial $p$.*

Most terminology for individual task-PIOAs can be carried over to task-PIOA families in a "pointwise" manner. For example, a task-PIOA family is said to be *closed* provided that every $\mathcal{T}_k$ is closed. Compatibility, parallel composition, and hiding for task-PIOA families are also defined pointwise. Results for composition and hiding for time-bounded task-PIOA families carry over easily from those for individual time-bounded task-PIOAs.

**Lemma 3.** *Suppose $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ are two compatible task-PIOA families, $\overline{\mathcal{T}}_1$ is $p_1$-time-bounded, and $\overline{\mathcal{T}}_2$ is $p_2$-time-bounded, where $p_1, p_2 : \mathsf{N} \to \mathsf{R}^{\geq 0}$. Then $\overline{\mathcal{T}}_1 \| \overline{\mathcal{T}}_2$ is a $c_{comp}(p_1 + p_2)$-time-bounded task-PIOA family.*

**Corollary 3.** *Suppose $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ are two compatible, polynomial-time-bounded task-PIOA families. Then $\overline{\mathcal{T}}_1 \| \overline{\mathcal{T}}_2$ is a polynomial-time-bounded task-PIOA family.*

**Lemma 4.** *Suppose $\overline{\mathcal{T}}$ is a $p$-time-bounded task-PIOA family, and $p'$ is a function, where $p, p' : \mathsf{N} \to \mathsf{R}^{\geq 0}$. Suppose that $\overline{S} = \{S_k\}_{k \in \mathsf{N}}$ is a family of sets of actions, where each $S_k$ is a $p'(k)$-time recognizable set of output actions for $\mathcal{T}_k$. Then $hide(\overline{\mathcal{T}}, \overline{S})$ is a $c_{hide}(p + p')$-time-bounded task-PIOA family.*

**Corollary 4.** *Suppose $\overline{\mathcal{T}}$ is a polynomial-time-bounded task-PIOA family. Suppose that $\overline{S} = \{S_k\}_{k \in \mathsf{N}}$ is a polynomial-time-recognizable family of sets of actions[9], where each $S_k$ is set of output actions for $\mathcal{T}_k$. Then $hide(\overline{\mathcal{T}}, \overline{S})$ is a polynomial-time-bounded task-PIOA family.*

Finally, we define time bounds for task schedule families.

**Definition 23.** *Given a task-PIOA family $\overline{\mathcal{T}}$ and a function $q : \mathsf{N} \to \mathsf{R}^{\geq 0}$, a family $\overline{\rho}$ of task schedules for $\overline{\mathcal{T}}$ is said to be* $q$-bounded *if $\rho_k$ is $q(k)$-bounded for every $k$.*

**Another way of expressing resource bounds:** In some of the computational security literature, resource bounds for machines are expressed, not in terms of a security parameter, but in terms of the lengths of machine inputs [Can01,BPW04b,Gol01,Küs06]. In fact, since the machines in question are interactive, the inputs that are considered for this purpose are sometimes allowed to arrive dynamically, during the execution of the protocol. This way of expressing resource bounds seems to be inspired by traditional Turing-machine-based complexity theory, wherein a Turing machine's resource bounds are expressed as functions of the lengths of its (initial) inputs. However, in interactive Turing machine settings, and in security settings in particular, this type of bound on the runtime becomes somewhat problematic. Thus, we express our bounds here simply in terms of the security parameter $k$.

### 4.3 Examples

For the rest of the paper, we fix a family $\overline{D} = \{D_k\}_{k \in \mathsf{N}}$ of finite domains; for concreteness, let $D_k$ be the set of bit strings of length $k$. Also, we fix a family $\overline{Tdp} = \{Tdp_k\}_{k \in \mathsf{N}}$ of sets of trap-door permutations such that the domain of every $f \in Tdp_k$ is $D_k$, and we define $\overline{Tdpp} = \{Tdpp_k\}_{k \in \mathsf{N}}$ to be the corresponding family of trap-door permutation pairs.

We model the OT protocol using a family of Transmitter automata, $\overline{Trans} = Trans(D_k, Tdp_k)_{k \in \mathsf{N}}$, and a family of Receiver automata, $\overline{Rec} = Rec(D_k, Tdp_k)_{k \in \mathsf{N}}$, where *Trans* and *Rec* are as defined in Sections 2.5 and 3.3. It is not hard to see that both $\overline{Trans}$ and $\overline{Rec}$ are polynomial-time-bounded.

[9] That is, there is a polynomial $p'$ such that, for every $k \in \mathsf{N}$, $S_k$ is a $p'(k)$-time-recognizable set.

### 4.4 An approximate implementation relation

In Section 3.4, we defined a *perfect implementation* relation, $\leq_0$, between two task-PIOAs. This correspondence is called "perfect" because it involves exact equality of trace distributions of the two task-PIOAs. However, for security protocols, it is also natural to consider *approximate implementation* relations. This is because small discrepancies between the behaviors of related systems may result from low-probability events such as an adversary guessing a secret key.

Here, we define a new approximate implementation relation $\leq_{\epsilon,p,q_1,q_2}$. This relation allows discrepancies in the correspondence, and also takes into account time bounds of the various automata involved. We begin by defining the *acceptance probability* for closed task-PIOAs that have a special *accept* output action:

**Definition 24.** *Let $\mathcal{T}$ be a closed task-PIOA with a special output action accept and let $\rho$ be a task schedule for $\mathcal{T}$. Then the* acceptance probability *for $\mathcal{T}$ and $\rho$ is defined to be:*

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

*where $\beta \leftarrow tdist(\mathcal{T}, \rho)$ means that $\beta$ is drawn randomly from $tdist(\mathcal{T}, \rho)$.*

That is, $Paccept(\mathcal{T}, \rho)$ is the probability that a trace chosen randomly from the trace distribution generated by $\rho$ contains the *accept* output action. From now on, we assume that every environment task-PIOA has *accept* as an output. The approximate implementation relation $\leq_{\epsilon,p,q_1,q_2}$ is defined as follows:

**Definition 25.** *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be comparable task-PIOAs and let $\epsilon, p \in \mathsf{R}^{\geq 0}$ and $q_1, q_2 \in \mathsf{N}$ be given. We define $\mathcal{T}_1 \leq_{\epsilon,p,q_1,q_2} \mathcal{T}_2$ as follows: given any $p$-time-bounded environment $\mathcal{E}$ for both $\mathcal{T}_1$ and $\mathcal{T}_2$, and any $q_1$-bounded task schedule $\rho_1$ for $\mathcal{T}_1 \| \mathcal{E}$, there is a $q_2$-bounded task schedule $\rho_2$ for $\mathcal{T}_2 \| \mathcal{E}$ such that $|Paccept(\mathcal{T}_1 \| \mathcal{E}, \rho_1) - Paccept(\mathcal{T}_2 \| \mathcal{E}, \rho_2)| \leq \epsilon$.*

In other words, from the perspective of a $p$-time-bounded environment, $\mathcal{T}_1$ and $\mathcal{T}_2$ "look almost the same," provided $\mathcal{T}_2$ can use at most $q_2$ steps to emulate $q_1$ steps of $\mathcal{T}_1$. The relation $\leq_{\epsilon,p,q_1,q_2}$ is transitive and preserved under composition and hiding, with certain adjustments to errors and time bounds.

**Lemma 5.** *Suppose $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ are three comparable task-PIOAs such that $\mathcal{T}_1 \leq_{\epsilon_{12},p,q_1,q_2} \mathcal{T}_2$ and $\mathcal{T}_2 \leq_{\epsilon_{23},p,q_2,q_3} \mathcal{T}_3$, where $\epsilon, p \in \mathsf{R}^{\geq 0}$ and $q_1, q_2, q_3 \in \mathsf{N}$. Then $\mathcal{T}_1 \leq_{\epsilon_{12}+\epsilon_{23},p,q_1,q_3} \mathcal{T}_3$.*

**Lemma 6.** *Suppose $\epsilon, p, p_3 \in \mathsf{R}^{\geq 0}$, and $q_1, q_2 \in \mathsf{N}$. Suppose that $\mathcal{T}_1$, $\mathcal{T}_2$ are comparable task-PIOAs such that $\mathcal{T}_1 \leq_{\epsilon,c_{comp}(p+p_3),q_1,q_2} \mathcal{T}_2$. Suppose that $\mathcal{T}_3$ is a $p_3$-time-bounded task-PIOA that is compatible with both $\mathcal{T}_1$ and $\mathcal{T}_2$. Then $\mathcal{T}_1 \| \mathcal{T}_3 \leq_{\epsilon,p,q_1,q_2} \mathcal{T}_2 \| \mathcal{T}_3$.*

**Lemma 7.** *Suppose $\epsilon, p \in \mathsf{R}^{\geq 0}$, and $q_1, q_2 \in \mathsf{N}$. Suppose that $\mathcal{T}_1$, $\mathcal{T}_2$ are comparable task-PIOAs such that $\mathcal{T}_1 \leq_{\epsilon,p,q_1,q_2} \mathcal{T}_2$. Suppose also that $S$ is a set of output actions of both $\mathcal{T}_1$ and $\mathcal{T}_2$. Then $hide(\mathcal{T}_1, S) \leq_{\epsilon,p,q_1,q_2} hide(\mathcal{T}_2, S)$.*

We extend the relation $\leq_{\epsilon,p,q_1,q_2}$ to task-PIOA families in the obvious way (i.e., pointwise).

**Definition 26.** *Let $\overline{\mathcal{T}}_1 = \{(\mathcal{T}_1)_k\}_{k\in\mathsf{N}}$ and $\overline{\mathcal{T}}_2 = \{(\mathcal{T}_2)_k\}_{k\in\mathsf{N}}$ be (pointwise)* comparable *task-PIOA families and let $\epsilon, p : \mathsf{N} \to \mathsf{R}^{\geq 0}$ and $q_1, q_2 : \mathsf{N} \to \mathsf{N}$ be given. We say that $\overline{\mathcal{T}}_1 \leq_{\epsilon,p,q_1,q_2} \overline{\mathcal{T}}_2$ provided that $(\mathcal{T}_1)_k \leq_{\epsilon(k),p(k),q_1(k),q_2(k)} (\mathcal{T}_2)_k$ for every $k$.*

The three preceding results carry over to task-PIOA families:

**Lemma 8.** *Suppose $\overline{\mathcal{T}}_1$, $\overline{\mathcal{T}}_2$ and $\overline{\mathcal{T}}_3$ are three comparable task-PIOA families such that $\overline{\mathcal{T}}_1 \leq_{\epsilon_{12},p,q_1,q_2} \overline{\mathcal{T}}_2$ and $\overline{\mathcal{T}}_2 \leq_{\epsilon_{23},p,q_2,q_3} \overline{\mathcal{T}}_3$, where $\epsilon_{12}, \epsilon_{23}, p : \mathsf{N} \to \mathsf{R}^{\geq 0}$ and $q_1, q_2 : \mathsf{N} \to \mathsf{N}$.*
*Then $\overline{\mathcal{T}}_1 \leq_{\epsilon_{12}+\epsilon_{23},p,q_1,q_3} \overline{\mathcal{T}}_3$.*

**Lemma 9.** *Suppose $\epsilon, p, p_3 : \mathsf{N} \to \mathsf{R}^{\geq 0}$, and $q_1, q_2 : \mathsf{N} \to \mathsf{N}$. Suppose $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ are comparable task-PIOA families such that $\overline{\mathcal{T}}_1 \leq_{\epsilon,c_{comp}(p+p_3),q_1,q_2} \overline{\mathcal{T}}_2$. Suppose that $\overline{\mathcal{T}}_3$ is a $p_3$-time-bounded task-PIOA family that is compatible with both $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$.*
*Then $\overline{\mathcal{T}}_1 \| \overline{\mathcal{T}}_3 \leq_{\epsilon,p,q_1,q_2} \overline{\mathcal{T}}_2 \| \overline{\mathcal{T}}_3$.*

**Lemma 10.** *Suppose $\epsilon, p : \mathsf{N} \to \mathsf{R}^{\geq 0}$, and $q_1, q_2 : \mathsf{N} \to \mathsf{N}$. Suppose that $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ are comparable task-PIOA families such that $\overline{\mathcal{T}}_1 \leq_{\epsilon,p,q_1,q_2} \overline{\mathcal{T}}_2$. Suppose also that $\overline{S}$ is a family of sets of output actions for both $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$.*
*Then $hide(\overline{\mathcal{T}}_1, \overline{S}) \leq_{\epsilon,p,q_1,q_2} hide(\overline{\mathcal{T}}_2, \overline{S})$.*

### 4.5 The relation $\leq_{neg,pt}$

Now we restrict attention to polynomial time bounds, for both individual steps and for schedule lengths, and to negligible error. This yields a generic version of approximate implementation, $\leq_{neg,pt}$. We use this relation in our analysis of security properties for Oblivious Transfer.

**Definition 27.** *A function $\epsilon : \mathsf{N} \to \mathsf{R}^{\geq 0}$ is said to be* negligible *if, for every constant $c \in \mathsf{R}^{\geq 0}$, there exists $k_0 \in \mathsf{N}$ such that $\epsilon(k) < \frac{1}{k^c}$ for all $k \geq k_0$. In other words, $\epsilon$ diminishes more quickly than the reciprocal of any polynomial.*

**Definition 28.** *Suppose $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ are comparable task-PIOA families. We say that $\overline{\mathcal{T}}_1 \leq_{neg,pt} \overline{\mathcal{T}}_2$ if, for all polynomials $p$ and $q_1$, there exist a polynomial $q_2$ and a negligible function $\epsilon$ such that $\overline{\mathcal{T}}_1 \leq_{\epsilon,p,q_1,q_2} \overline{\mathcal{T}}_2$.*

We show that $\leq_{neg,pt}$ is transitive and preserved under composition and hiding; for composition, we need to assume polynomial time bounds for one of the task-PIOA families.

**Theorem 4.** *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 $\overline{\mathcal{T}}_1 \leq_{neg,pt} \overline{\mathcal{T}}_3$.*

**Theorem 5.** *Suppose $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ are comparable task-PIOA families such that $\overline{\mathcal{T}}_1 \leq_{neg,pt} \overline{\mathcal{T}}_2$, and suppose $\overline{\mathcal{T}}_3$ is a polynomial-time-bounded task-PIOA family that is compatible with both $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$. Then $\overline{\mathcal{T}}_1 \| \overline{\mathcal{T}}_3 \leq_{neg,pt} \overline{\mathcal{T}}_2 \| \overline{\mathcal{T}}_3$.*

**Theorem 6.** *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 $\overline{S}$ is a family of sets of output actions for both $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$.*
*Then $hide(\overline{\mathcal{T}}_1, \overline{S}) \leq_{neg,pt} hide(\overline{\mathcal{T}}_2, \overline{S})$.*

### 4.6 Simulation relations, revisited

In Section 3.5, we defined simulation relations for ordinary task-PIOAs, without resource bounds. These simulation relations were shown to be sound for proving that one task-PIOA *perfectly implements* another, according to the relation $\leq_0$. However, we would like also to use these simulation relations in a setting with time bounds, and in particular, to show that one time-bounded task-PIOA implements another according to the relation $\leq_{neg,pt}$. In order to do this, we require an additional assumption about the lengths of matching task schedules:

**Definition 29.** *Suppose that $R$ is a simulation relation from $\mathcal{T}_1$ to $\mathcal{T}_2$ using task mapping $c$, and $b \in \mathsf{N}$. Then $R$ is said to be* $b$-bounded *if $c(\rho_1, T)$ is $b$-bounded for every $\rho_1$ and $T$.*

Then we have the following theorem:

**Theorem 7.** *Suppose that $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ are comparable closed task-PIOA families, $b \in \mathsf{N}$. Suppose that for every polynomial $p$, every $k$, and every $p(k)$-bounded environment $\mathcal{E}_k$ for $(\overline{\mathcal{T}}_1)_k$ and $(\overline{\mathcal{T}}_2)_k$, there exists a $b$-bounded simulation relation $R_k$ from $(\overline{\mathcal{T}}_1)_k \| \mathcal{E}_k$ to $(\overline{\mathcal{T}}_2)_k \| \mathcal{E}_k$. Then $\overline{\mathcal{T}}_1 \leq_{neg,pt} \overline{\mathcal{T}}_2$.*

**Proof.** In [CCK+06b], soundness of simulation relations is proved as follows. Given a simulation relation $R$ from closed task-PIOA $\mathcal{T}_1$ to closed task-PIOA $\mathcal{T}_2$, and a task schedule $\rho_1 = T_1, T_2, \ldots$ for $\mathcal{T}_1$, we construct a task schedule $\rho_2$ for $\mathcal{T}_2$ by concatenating sequences returned by $c$; that is,

$$\rho_2 := corr(\lambda, T_1) \ldots corr(T_1 \ldots T_n, T_{n+1}) \ldots$$

We then prove that $tdist(\rho_1) = tdist(\rho_2)$. Note that, if $R$ is $b$-bounded, then the length of $\rho_2$ is at most $b \cdot |\rho_1|$.

Now let polynomials $p$ and $q_1$ be given as in the definition of $\leq_{neg,pt}$. Let $q_2$ be $b \cdot q_1$ and $\epsilon$ be the constant-0 function. Using the proof outlined above, it is easy to check that $q_2$ and $\epsilon$ satisfy the requirements for $\leq_{neg,pt}$. □

## 5 Hard-Core Predicates

In this section, we discuss a well-known concept in cryptography—a *hard-core predicate* for trap-door functions—that we use in our Oblivious Transfer protocol

model and proof. The standard definition of a hard-core predicate involves a comparison between the results of two probabilistic experiments. We reformulate this definition in terms of time-bounded task-PIOAs, using our approximate implementation relation $\leq_{neg,pt}$, and show that the reformulation is equivalent to the standard definition. Then, to demonstrate how our new definition can be used in security protocol proofs, we present a simple example, showing that a hard-core predicate retains its security properties if it is used twice. This example is derived from [CCK+06c], where we used it in the proof for the corrupted-Transmitter case. In Section 7, we will describe how our new definition is used in the corrupted-Receiver case of our latest OT proof [CCK+05].

### 5.1 Definitions

In the traditional definition, a function $B : \bigcup_{k\in\mathbf{N}} D_k \rightarrow \{0,1\}$ is said to be a hard-core predicate for a trap-door permutation family $\overline{Tdp}$ if, whenever $f$ and $z$ are chosen randomly from $Tdp_k$ and $D_k$, respectively, the bit $B(f^{-1}(z))$ "appears random" to a probabilistic-polynomial-time observer, even if $f$ and $z$ are given to the observer as inputs. This captures the idea that $f^{-1}(z)$ cannot be computed efficiently from $f$ and $z$. The following is a slight reformulation of Definition 2.5.1 of [Gol01].

**Definition 30.** *A* hard-core predicate *for $\overline{D}$ and $\overline{Tdp}$ is a predicate $B : \bigcup_{k\in\mathbf{N}} D_k \rightarrow \{0,1\}$, such that (1) $B$ is polynomial-time computable and (2) For every probabilistic polynomial-time non-uniform predicate $G = \{G_k\}_{k\in\mathbf{N}}$,*[10] *there is a negligible function $\epsilon$ such that, for all $k$,*

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

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

Our reformulated definition uses $\leq_{neg,pt}$ to express the idea that $B(f^{-1}(z))$ "appears random". We define two Task-PIOA families, $\overline{SH}$ (for "System providing a Hard-core bit") and $\overline{SHR}$ (for "System in which the Hard-core bit is replaced by a Random bit"). The former outputs random elements $f$ and $z$ from $Tdp_k$ and $D_k$, and the bit $B(f^{-1}(z))$. The latter does the same except $B(f^{-1}(z))$ is replaced by a random element from $\{0,1\}$. Then $B$ is said to be a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ if $\overline{SH} \leq_{neg,pt} \overline{SHR}$.

[10] 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 $p(k)$.

**Definition 31.** *$B$ is said to be a* hard-core predicate *for $\overline{D}$ and $\overline{Tdp}$ if $\overline{SH} \leq_{neg,pt} \overline{SHR}$, where $\overline{SH}$ and $\overline{SHR}$ are defined as follows. For each $k \in \mathsf{N}$, let $\mu_k$ and $\mu'_k$ denote the uniform distributions on $Tdp_k$ and $D_k$, respectively. Let $\mu''$ be the uniform distribution on $\{0,1\}$.*

*Then $\overline{SH}$ is defined to be $hide(\overline{Src_{tdp}} \| \overline{Src_{yval}} \| \overline{H}, \{rand_{yval}(*)\})$, where*

1. *$\overline{Src_{tdp}}$ is the family $\{Src_{tdp}(Tdp_k, \mu_k)\}_{k \in \mathsf{N}}$;*
2. *$\overline{Src_{yval}}$ is the family $\{Src_{yval}(D_k, \mu'_k)\}_{k \in \mathsf{N}}$; and*
3. *each $H_k$ obtains the permutation $f$ from $Src_{tdp}(Tdp_k, \mu_k)$ and the element $y \in D_k$ from $Src_{yval}(D_k, \mu'_k)$, and outputs $z := f(y)$ via action $rand_{zval}$ and $b := B(y)$ via action $rand_{bval}$.*

*Since $f$ is a permutation, this is equivalent to choosing $z$ randomly and computing $y$ as $f^{-1}(z)$. Each $H_k$ is defined formally as $H(D_k, Tdp_k, B)$, where $H(D, Tdp, B)$ is defined in Figure 5.*

*Also, $\overline{SHR}$ is defined to be $\overline{Src_{tdp}} \| \overline{Src_{zval}} \| \overline{Src_{bval}}$, where*

1. *$\overline{Src_{tdp}}$ is as in $\overline{SH}$;*
2. *$\overline{Src_{zval}} = \{Src_{zval}(D_k, \mu'_k)\}_{k \in \mathsf{N}}$; and*
3. *$\overline{Src_{bval}} = \{Src_{bval}(\{0,1\}, \mu'')_k\}_{k \in \mathsf{N}}$.*

These two systems are depicted in Figure 6. There, the automata labeled with "$\mathfrak{S}$" represent the random source automata. Observe that this definition makes use of the asymmetry of the $\leq_{neg,pt}$ relation, as it is not the case that $\overline{SHR} \leq_{neg,pt} \overline{SH}$. For example, $\overline{SH}$ cannot execute a $rand_{tdp}(*)$ action before executing the $rand_{zval}(*)$ action, whereas this may happen in $\overline{SHR}$.

We claim that these two definitions of hard-core bits are equivalent.

**Theorem 8.** *$B$ is a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ according to Definition 30 if and only if it is a hard-core predicate for $\overline{D}$ and $\overline{Tdp}$ according to Definition 31.*

The proof of Theorem 8 is rather technical and appears in [CCK+05].

### 5.2 Example

Our new definition of hard-core predicates, Definition 31, seems to be well suited for use in modeling and analyzing security protocols. For a simple example to illustrate how this definition can be exploited, we show here that a hard-core predicate can be applied twice, and a probabilistic polynomial-time environment still cannot distinguish the resulting outputs from random values. We used this fact in the earlier version of our OT proof [CCK+06c], in a situation where 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 permutation.

We show that, 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

$H(D, Tdp, B)$ :

**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:
- $fix - bval$
- $fix - zval$

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

**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:
- $fval, yval \neq \perp$
- $zval = \perp$

Effect:
- $zval := fval(yval)$

$fix - bval$
Precondition:
- $yval \neq \perp$
- $bval = \perp$

Effect:
- $bval := B(yval)$

$rand(z)_{zval}$
Precondition:
- $z = zval$

Effect:
- none

$rand(b)_{bval}$
Precondition:
- $b = bval$

Effect:
- none

**Fig. 5.** Hard-core predicate automaton, $H(D, Tdp, B)$

randomly-chosen trap-door permutation, $z(0)$ and $z(1)$ are randomly-chosen elements of $D_k$, and $b(0)$ and $b(1)$ are randomly-chosen bits. We do this by defining two task-PIOA families, $\overline{SH2}$ and $\overline{SHR2}$, that produce the two distributions, and showing that $\overline{SH2} \leq_{neg,pt} \overline{SHR2}$. Task-PIOA family $\overline{SH2}$ is defined as

$$hide(\overline{Src_{tdp}} \| \overline{Src_{yval0}} \| \overline{Src_{yval1}} \| \overline{H0} \| \overline{H1}, \{rand(*)_{yval0}, rand(*)_{yval1}\}),$$

where $\overline{Src_{tdp}}$ is as in the definition of $\overline{SH}$, $\overline{Src_{yval0}}$ and $\overline{Src_{yval1}}$ are isomorphic to $\overline{Src_{yval}}$ in $\overline{SH}$, and $\overline{H0}$ and $\overline{H1}$ are two instances of $\overline{H}$ (with appropriate

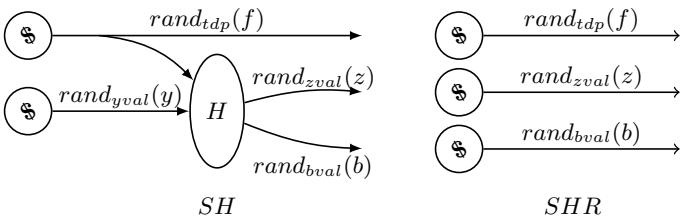


**Fig. 6.** $SH$ and $SHR$

renaming of actions). Task-PIOA family $\overline{SHR2}$ is defined as

$$(\overline{Src_{tdp}} \| \overline{Src_{zval0}} \| \overline{Src_{zval1}} \| \overline{Src_{bval0}} \| \overline{Src_{bval1}}),$$

where $\overline{Src_{tdp}}$ is as in $\overline{SH2}$, $\overline{Src_{zval0}}$ and $\overline{Src_{zval1}}$ are isomorphic to $\overline{Src_{zval}}$ in $\overline{SHR}$, and $\overline{Src_{bval0}}$ and $\overline{Src_{bval1}}$ are isomorphic to $\overline{Src_{bval}}$ in $\overline{SHR}$.

**Theorem 9.** *If $B$ is a hard-core predicate, then $\overline{SH2} \leq_{neg,pt} \overline{SHR2}$.*

**Proof.** Theorem 8 implies that $\overline{SH} \leq_{neg,pt} \overline{SHR}$. To prove that $\overline{SH2} \leq_{neg,pt} \overline{SHR2}$, we introduce a new task-PIOA family $\overline{Int}$, which is intermediate between $\overline{SH2}$ and $\overline{SHR2}$. $\overline{Int}$ is defined as

$$hide(\overline{Src_{tdp}} \| \overline{Src_{yval0}} \| \overline{H0} \| \overline{Src_{zval1}} \| \overline{Src_{bval1}}, \{rand(*)_{yval0}\}),$$

where $\overline{Src_{tdp}}$ is exactly as in $\overline{SH2}$ and $\overline{SHR2}$; $\overline{Src_{yval0}}$ and $\overline{H0}$ are as in $\overline{SH2}$; and $\overline{Src_{zval1}}$ and $\overline{Src_{bval1}}$ are as in $\overline{SHR2}$. Thus, $\overline{Int}$ generates $bval0$ using the hard-core predicate $B$, as in $\overline{SH2}$, and generates $bval1$ randomly, as in $\overline{SHR2}$.

To see that $\overline{SH2} \leq_{neg,pt} \overline{Int}$, note that Definition 31 implies that

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

because these two systems are simple renamings of $\overline{SH}$ and $\overline{SHR}$. Now let $\overline{I}$ be the task-PIOA family $hide(\overline{Src_{yval0}} \| \overline{H0}, \{rand(*)_{yval0}\}$. It is easy to see, from the code for the two components of $\overline{I}$, that $\overline{I}$ is polynomial-time-bounded. Then by Theorem 5,

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

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

Similarly, $\overline{Int} \leq_{neg,pt} \overline{SHR2}$. Since $\overline{SH2} \leq_{neg,pt} \overline{Int}$ and $\overline{Int} \leq_{neg,pt} \overline{SHR2}$, transitivity of $\leq_{neg,pt}$ (Theorem 4) implies that $\overline{SH2} \leq_{neg,pt} \overline{SHR2}$. □

The proof of Theorem 9 differs from corresponding proofs usually presented in the computational cryptography community. A traditional proof proceeds by contradiction, showing that, if an adversary can distinguish $\overline{SH2}$ from $\overline{SHR2}$, then a related adversary can distinguish $\overline{SH}$ from $\overline{SHR}$, which contradicts a

computational assumption. Here we use a direct argument: we exhibit an interface (which plays the role of the reduction in the traditional approach), and we use the composition property of the $\leq_{neg,pt}$ relation. We believe that such positive arguments provide more insight into the way the different assumptions are used.

## 6 Computational Security

So far in the paper, we have defined time-bounded task-PIOAs, our basic model for resource-bounded concurrent computation. We have illustrated their use in modeling hard-core predicates for trap-door functions. In this section, we explain how we use time-bounded task-PIOAs to define the security of cryptographic protocols, illustrating with our Oblivious Transfer example.

Our method follows the general outline of [Can01,PW01], which are standard references for simulation-based security in the computational cryptography community. We first specify the *functionality* that the protocol is supposed to realize. Then we define the *protocol*, which consists of protocol parties and auxiliary services, and the class of *adversaries* with which the protocol must contend. Together, the protocol and an adversary comprise the *Real System*. Finally, we define what it means for a protocol to *securely realize* its specified functionality.

### 6.1 The functionality

We represent the *functionality* for a security protocol as a single task-PIOA. It represents a "trusted party" that receives protocol inputs and returns protocol outputs at endpoints corresponding to the protocol parties. See [Can01] for many examples of classical cryptographic functionalities.

For OT, the functionality we use is the task-PIOA *Funct* given in Sections 2.5 and 3.3. Recall that it behaves as follows: It waits for two bits $(x(0), x(1))$ representing the inputs for the *Transmitter*, and one bit $i$ representing the input for the *Receiver*. Then it outputs the bit $x(i)$ at the Receiver end, and nothing at the Transmitter end.

This definition of *Funct* is designed for the special case in which neither party is corrupted. For the case we prove in [CCK+05], in which only the Receiver is corrupted, *Funct* must be modified slightly, by renaming the $out(w)_{Rec}$ outputs to $out'(w)_{Rec}$. The prime symbol is added here so that, instead of passing directly from *Funct* to the external environment, the output can pass indirectly, via an adversarial component called the *simulator*. We will discuss the simulator in Section 6.4.

The only other interesting case of the proof is the one in which only the Transmitter is corrupted. For this case, *Funct* would again have to be redefined. In particular, it would need additional input and output actions for synchronizing with the simulator, so that all possible trace distributions generated by the protocol can indeed be simulated. It remains to carry out the detailed analysis for this case; so far, we have analyzed it only for our more restrictive, earlier task-PIOA model [CCK+06c].

### 6.2 The protocol

The protocol consists of task-PIOAs representing protocol parties and any auxiliary services they require, such as random sources. Since the definitions of protocols are typically parameterized by a security parameter $k$, we define a protocol as a task-PIOA family $\overline{\pi} = \{\pi_k\}_{k \in \mathsf{N}}$, where $\pi_k$ is the composition of task-PIOAs representing the different protocol parties and auxiliary services for parameter $k$.

For Oblivious Transfer, the protocol parties for $k$ are $Trans(D_k, Tdp_k)$ and $Rec(D_k, Tdp_k)$, as discussed in Section 4.3. Also $\pi_k$ includes two random source automata: $Src_{pval}(Tdpp_k, \mu_k)$, which provides a random pair of $D_k$ values to $Rec(D_k, Tdp_k)$, and $Src_{yval}(D_k \times D_k, \nu_k)$, which provides a random trap-door permutation pair to $Trans(D_k, Tdp_k)$. Here $\mu_k$ is the uniform distribution on $Tdpp_k$ and $\nu_k$ is the uniform distribution on $D_k \times D_k$.

Recall that the protocol executes as follows. First *Trans* selects a random trap-door permutation $f$ from $Tdp_k$, together with its inverse $f^{-1}$, and sends $f$ to *Rec*. Then, using its input bit $i$ and two randomly selected elements $(y(0), y(1))$ of $D_k$, *Rec* computes the pair $(z(0), z(1)) = (f^{1-i}(y(0)), f^i(y(1)))$, and sends it to *Trans*. Finally, using its input bits $(x(0), x(1))$, *Trans* computes the pair $(b(0), b(1)) = (x(0) \oplus B(f^{-1}(z(0))), x(1) \oplus B(f^{-1}(z(1))))$ and sends it to *Rec*, who can now recover $x(i)$ as $B(y(i)) \oplus b(i)$.

### 6.3 The adversary

To analyze the security of a protocol $\overline{\pi}$, we consider $\overline{\pi}$ in combination with an *adversarial communication service*. Depending on the context, adversaries may have different capabilities: they may have passive or active access to the network, may be able to corrupt parties (either statically or dynamically), may assume partial or full control of the parties, etc. Various scenarios are discussed in [Can01]. We specify a particular class of adversaries by defining appropriate restrictions on the signature and transition relation of adversary task-PIOAs. By composing an adversary task-PIOA $Adv_k$ with a protocol task-PIOA $\pi_k$, we obtain a *real system*, $RS_k$.

For the OT protocol, we consider polynomial-time-bounded families of adversaries. The adversaries have passive access to protocol messages: they receive and deliver messages (possibly delaying, reordering, duplicating, or losing them), but do not modify messages or compose new ones. They may corrupt parties only statically (at the start of execution). They are "honest-but-curious", which means that they obtain access to all internal information of the corrupted parties, but the parties continue to follow the protocol definition.

In this paper and in [CCK+05], we discuss only one case, in which only the Receiver is corrupted. In this case, the adversary gains access to the input and output of *Rec* (that is, $i$ and $x(i)$), and to its random choices (that is, $y(0)$ and $y(1)$). However, as noted above, *Rec* continues to follow the protocol definition, so we model it formally as a component distinct from the adversary.

In somewhat more detail, the adversary $Adv_k$ acts as a message system connecting $Trans_k$ and $Rec_k$: it has inputs corresponding to the *send* actions of $Trans_k$ and $Rec_k$, and outputs corresponding to the *receive* actions. $Adv_k$ does not corrupt or compose messages, but may delay, reorder, duplicate, or lose them. In addition, because the Receiver is corrupted, $Adv_k$ has inputs of the form $in(i)_{Rec}$ by which it overhears the Receiver's inputs. $Adv_k$ also has inputs of the form $out'(w)_{Rec}$ and corresponding outputs of the form $out(w)_{Rec}$; using these, $Adv_k$ acts as a *relay*, receiving output information at the Receiver end and conveying it, with possible delays, to the external environment.[11] $Adv_k$ also overhears the outputs of $Src_{yval}$, which provides the Receiver's random choice of $y$. Finally, $Adv_k$ may have arbitrary other input, output, and internal actions.

The components of Real System $RS_k$ and their interactions are depicted in Figure 7. Incoming and outgoing arrows at the top of the diagram are assumed to connect to an external environment. Notice, in particular, the use of the primes for relaying outputs: the $out'(w)_{Rec}$ actions are outputs of $Rec_k$ and inputs of $Adv_k$, and the $out(w)_{Rec}$ actions are outputs of $Adv_k$ to the external environment. Formally, these components are all combined using the composition operation for task-PIOAs, and then all output actions except the ones at the environment interface are hidden.

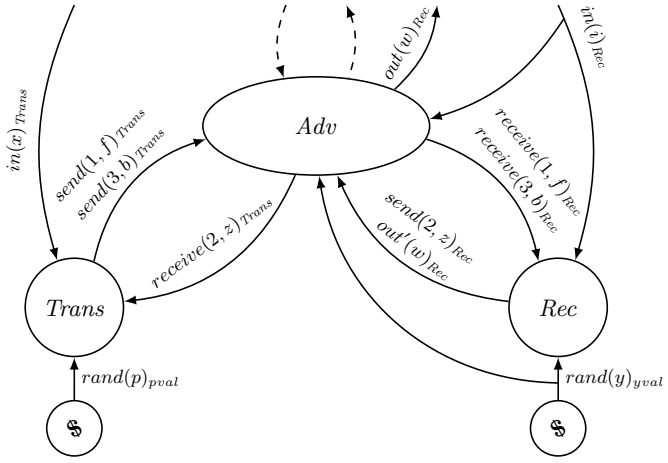


**Fig. 7.** *RS* when only Receiver corrupted

[11] Actually, we could have allowed the adversary to relay Receiver inputs also, rather than simply overhearing them. This modeling approach was taken, for example, in [BPW04b].

**Adversarial scheduling:** Recall our remark about adversarial scheduling in Section 3.1. The adversaries we consider for OT are allowed to determine their next moves adaptively, based on their view of the computation so far. In particular, $Adv_k$ may determine the order in which it delivers messages, based on the messages that have been sent so far, and based on Receiver inputs, outputs, and random choices. Similarly, based on the same information, $Adv_k$ may determine what additional internal and output actions it performs, and when.

### 6.4 Computational security

In order to prove that a protocol securely realizes a functionality, we show that, for every adversary family $\overline{Adv}$ of the considered class, there is another task-PIOA family $\overline{Sim}$, called a *simulator family*, that can mimic the behavior of $\overline{Adv}$ by interacting only with the functionality. This implies that the protocol does not reveal to the adversary any information that it does not need to reveal—that is, anything not revealed by the functionality itself.

The quality of emulation is evaluated from the viewpoint of the *environment*, which is also a task-PIOA family. Thus, security of the protocol says that no environment can efficiently decide whether it is interacting with the real system (i.e., the composition of the protocol and the adversarial communication system) or with the composition of the functionality and the simulator. We call this latter composition the *ideal system*. We formalize this indistinguishability condition by the following definition.

**Definition 32.** *Let $\overline{\pi}$, $F$ and $\mathcal{A}$ be a protocol family, a functionality and a class of adversary families, respectively. Let $\overline{F}$ denote the family in which every $F_k$ equals $F$. Then we say that $\overline{\pi}$* securely emulates *$F$ with respect to $\mathcal{A}$ provided that the following holds: for every real-system family $\overline{RS}$ (constructed from $\overline{\pi}$ and some polynomial-time-bounded adversary family $\overline{Adv} \in \mathcal{A}$), there exists an ideal-system family $\overline{IS}$ (constructed from $\overline{F}$ and some polynomial-time-bounded simulator family $\overline{Sim}$) such that $\overline{RS} \leq_{neg,pt} \overline{IS}$.*

In Definition 32, quantification over environments is encapsulated within the definition of $\leq_{neg,pt}$: $\overline{RS} \leq_{neg,pt} \overline{IS}$ says that, for every polynomial time-bounded environment family $\overline{Env}$ and every polynomial-bounded task schedule for $\overline{RS}\|\overline{Env}$, there is a polynomial-bounded task schedule for $\overline{IS}\|\overline{Env}$ such that the acceptance probabilities in these two systems differ by a negligible amount.

For Oblivious Transfer, the components of Ideal System $IS_k$ and their interactions are depicted in Figure 8. Incoming and outgoing arrows at the top of the diagram are assumed to connect to the environment. Again, these components are combined using the composition operation for task-PIOAs, and then all output actions except the ones at the environment interface are hidden.

Based on the definition of secure emulation, correctness theorems for security protocols take the following form:

**Theorem 10.** *Protocol family $\overline{\pi}$ securely emulates functionality $F$ with respect to adversary family $\mathcal{A}$.*

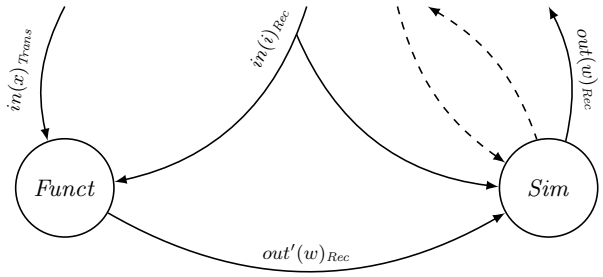


**Fig. 8.** $IS$ when only Receiver corrupted

Theorem 10 expresses both functional correctness and security. Functional correctness is captured by the requirement that the inputs and (direct and relayed) outputs between the protocol and the environment in the real system execution are consistent with those allowed by the functionality. For Oblivious Transfer, this means that, in a real system execution, the inputs $in(x)_{Trans}$ and $in(i)_{Rec}$, as well as the output $out(w)_{Rec}$, are consistent with those allowed by the functionality; in other words, if an output bit is produced at the Receiver endpoint, then it is the correct bit. Security is captured by the fact that the adversary's interactions with the environment via its additional inputs and outputs can be emulated successfully by a simulator, who interacts only with the functionality. For Oblivious Transfer, this means that the adversary is unable to output the non-chosen bit, $x(1-i)$, from its interaction with the real system, except with negligible probability.

We are currently proving composition theorems for secure emulation; that is, if we are given a collection of protocols, each of which securely emulates a corresponding functionality, then the composition of the protocols securely emulates the composition of the functionalities. Our proofs are based on composition theorems for the $\leq_{neg,pt}$ relation on time-bounded task-PIOAs. Similar theorems have been proved in other frameworks [Can01,BPW04a,MMS03].

## 7 Levels-of-Abstraction Proofs

In order to prove Theorem 10 for the OT protocol and the adversaries of Section 6, we must define an ideal-system family $\overline{IS}$ such that $\overline{RS} \leq_{neg,pt} \overline{IS}$. To that end, we build a structured simulator family $\overline{SSim}$ from any adversary family $\overline{Adv}$. We define the needed ideal-system family as $\overline{SIS}$, where $SIS_k = Funct \| SSim_k$.

To show that $\overline{RS} \leq_{neg,pt} \overline{SIS}$, we define two intermediate system families, $\overline{Int1}$ and $\overline{Int2}$, and decompose the proof into three subgoals: $\overline{RS} \leq_{neg,pt} \overline{Int1}$, $\overline{Int1} \leq_{neg,pt} \overline{Int2}$, and $\overline{Int2} \leq_{neg,pt} \overline{SIS}$. Transitivity of $\leq_{neg,pt}$ (Theorem 4) then yields $\overline{RS} \leq_{neg,pt} \overline{SIS}$, as needed. All arguments involving computational indistinguishability and other cryptographic issues are isolated within the proof

of the second subgoal. The other two subgoals are proved using simulation relations.

### 7.1 Simulator strategy

We construct the structured simulator family $\overline{SSim}$ from the given adversary family $\overline{Adv}$, following the standard informal construction used in computational cryptography [CLOS02,Gol04]. For every index $k$, $SSim_k$ is the composition of $Adv_k$ with an abstract version of $\pi_k$ based on a task-PIOA $TR(D_k, Tdp_k)$. $TR(D_k, Tdp_k)$ works as follows: First, it selects and sends a random element $f \in Tdp_k$, as *Trans* would. Then, after $Adv_k$ has delivered $f$, $TR$ emulates *Rec*: it chooses a random pair $(y(0), y(1))$ of elements of $D_k$, and sends a round 2 message containing $(f^{1-i}(y(0)), f^i(y(1)))$. Next, $TR$ computes the pair $(b(0), b(1))$ and sends it in a round 3 message. Specifically, $TR$ computes $b(i)$ as $B(y(i)) \oplus x(i)$ and $b(1-i)$ as a random bit. Note that this requires that $TR$ learn $x(i)$, which is the chosen Transmitter input. Although $TR$ does not have direct access to this input, $TR$ obtains $x(i)$ as output from *Funct* at the Receiver end. Detailed code for $TR(D, Tdp)$ appears in Figures 10 and 11. Figure 9 shows the components of $SIS$ and their interactions.

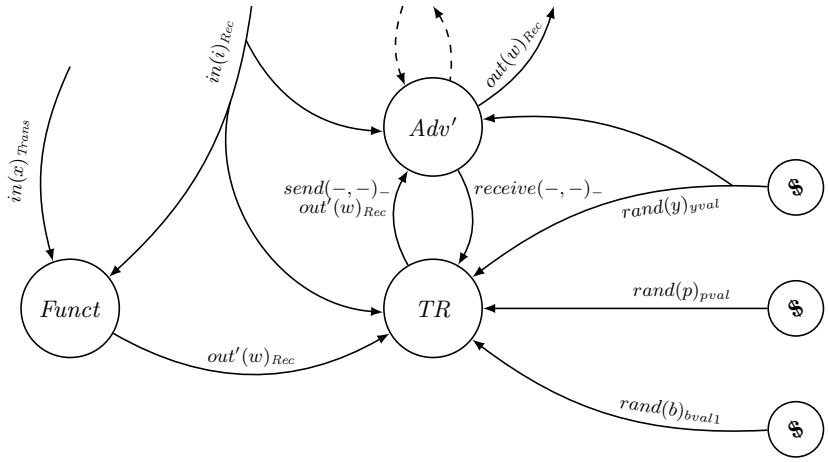


**Fig. 9.** $SIS$ when only Receiver corrupted

Observe that, if $\overline{Adv}$ is polynomial-time-bounded, then so is $\overline{SSim}$. Also, if we define $\overline{SIS}$ by $SIS_k = Funct \| SSim_k$, then $\overline{SIS}$ is an ideal system family.

### 7.2 Intermediate systems

The $Int1_k$ system is the same as $SIS_k$, except that $TR$ is replaced by $TR1$, which differs from $TR$ as follows: (1) It has an extra input $in(x)_{Trans}$, which provides

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

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

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

**State:**
$ival, wval \in \{0,1,\perp\}$, initially $\perp$
$pval \in Tdpp \cup \{\perp\}$, initially $\perp$
$yval, zval \in (\{0,1\} \to D) \cup \{\perp\}$, initially $\perp$
$bval1 \in \{0,1,\perp\}$, initially $\perp$
$bval \in (\{0,1\} \to \{0,1\}) \cup \{\perp\}$, initially $\perp$
$received \subseteq \{1,2,3\}$, initially $\emptyset$

**Fig. 10.** Code for $TR(D, Tdp)$, Part I

the Transmitter input. (2) It computes the round 3 message differently: the bit $b(i)$ is computed as in $TR$, but the bit $b(1-i)$ is computed using the hard-core predicate $B$, as $B(f^{-1}(z(1-i))) \oplus x(1-i)$. The $Int2_k$ system is the same as $SIS_k$ except that it includes a random source automaton $Src_{cval1}$ that produces a random bit $cval1$, and also $TR$ is replaced by $TR2$, which is essentially the same as $TR1$ except that $b(1-i)$ is computed as $cval1 \oplus x(1-i)$.

### 7.3 Simulation relation proofs

Our proofs that $\overline{RS} \leq_{neg,pt} \overline{Int1}$ and $\overline{Int2} \leq_{neg,pt} \overline{SIS}$ use simulation relations, specifically, Theorem 7.

To prove that $\overline{RS} \leq_{neg,pt} \overline{Int1}$, we fix any polynomial $p$, any $k \in \mathsf{N}$, and any $p(k)$-bounded environment $Env_k$ for $RS_k$ and $Int1_k$. We show that there is a 2-bounded simulation relation $R_k$ from $RS_k \| Env_k$ to $Int1_k \| Env_k$. Then Theorem 7 implies that $\overline{RS} \leq_{neg,pt} \overline{Int1}$.

The simulation relation $R_k$ relates states to states, rather than measures on executions to measures on executions. Thus, it does not utilize the full generality of our new simulation relation notion—Segala's original simulation relation would be sufficient here. Relation $R_k$ is defined in terms of a list of simple equations between state variables of $RS_k \| Env_k$ and $Int1_k \| Env_k$. In particular, $R_k$ is the identity on the states of $Adv_k$ and $Env_k$.

**Transitions:**

$out'(w)_{Rec}$
Effect:
  if $wval = \perp$ then $wval := w$

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

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

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

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

$receive(1, f)_{Rec}$
Effect:
  $received := received \cup \{1\}$

$receive(2, z)_{Trans}$
Effect:
  $received := received \cup \{2\}$

$receive(3, b)_{Rec}$
Effect:
  if $ival, yval \neq \perp$ then
    $received := received \cup \{3\}$

$fix - zval_{Rec}$
Precondition:
  $ival, pval, yval \neq \perp$
  $1 \in received$
  $zval = \perp$
Effect:
  $zval(ival) := pval.funct(yval(ival))$
  $zval(1 - ival) := yval(1 - ival)$

$fix - bval_{Trans}$
Precondition:
  $ival, wval, yval, zval, bval1 \neq \perp$
  $2 \in received$
  $bval = \perp$
Effect:
  $bval(ival)) := B(yval(ival)) \oplus wval$
  $bval(1 - ival) := bval1$

$out''(w)_{Rec}$
Precondition:
  $w = wval$
  $3 \in received$
Effect:
  none

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

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

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

**Fig. 11.** Code for $TR(D, Tdp)$, Part II

To show that $R_k$ is in fact a simulation relation, we first define the $c$ mapping between task schedules. All the tasks in $RS_k \| Env_k$ correspond to essentially the same tasks in $Int1_k \| Env_k$, with one exception: the $fix-bval$ task in $RS_k \| Env_k$, by which *Trans* in the $RS$ system determines the value of $bval$, having already received its own input and a round 2 message. This task is mapped to a sequence of two tasks in $Int1_k \| Env_k$: the $out'$ task, by which *Funct* outputs its result to the simulator component in the *Int1* system, followed by the $fix - bval$ task of that simulator. This correspondence reflects the fact that the simulator in *Int1* does not have access to the Transmitter input, as *Trans* does in $RS$; instead, it obtains the needed portion of that input—the chosen bit—as output from *Funct*.

Actually verifying the step condition, once we have $R_k$ and the $c$ mapping, is a long and tedious, but mostly straightforward case analysis. One point worth noting is how the expansion operation is used in the proof. Consider, for example, a step involving communication between $Adv_k$ and $Env_k$. As a result of this step, either or both of these components may choose their next states probabilistically. Since the state of $Adv_k$ is assumed to be the same in both systems before the step, and likewise for $Env_k$, the same probability distribution of new states for these two components results when the step is performed in the two systems. Use of expansion allows us to split up the resulting probability distributions, by relating states in the two systems in which the choices result in the same outcome. Technically, this expansion is formalized using Corollary 2, where the index set $I$ represents the set of possible outcomes of the random choices of $Adv_k$ and $Env_k$.

To prove that $\overline{Int2} \leq_{neg,pt} \overline{SIS}$, we again fix $p$, $k$, and $Env_k$. This time we show that there is a 1-bounded simulation relation $S_k$ from $Int2_k \| Env_k$ to $SIS_k \| Env_k$. In this case, the only difference between the two systems is that $b(1-i)$, the non-selected bit, is chosen randomly in $SIS_k$ (using $bval1$), but is defined as the $\oplus$ of a random bit ($cval1$) and the actual Transmitter input bit, in $Int2_k$. Informally speaking, this difference should not matter, since the resulting bit is randomly chosen in both cases. (Notice, this shows our simulation relations can be used to prove the security of the one-time pad.)

Formally, the simulation relation $S_k$ has a somewhat more complicated structure than $R_k$, relating probability measures on states in the two systems. In addition to a list of simple equations between state variables, $S_k$ includes conditions relating a measure on values for $cval1$ to one for $bval1$.

### 7.4 Computational indistinguishability proof

The middle stage of our proof, showing that $\overline{Int1} \leq_{neg,pt} \overline{Int2}$, uses a computational argument based on our definition of a hard-core predicate in Section 5. The only difference between $\overline{Int1}$ and $\overline{Int2}$ is that a single use of $B(f^{-1}(z(1-i)))$ in $\overline{Int1}$ is replaced by a random bit in $\overline{Int2}$. But this is precisely the difference between the $\overline{SH}$ and $\overline{SHR}$ systems discussed in Section 5. Definition 31 says that $\overline{SH} \leq_{neg,pt} \overline{SHR}$; now we exploit this fact in showing that $\overline{Int1} \leq_{neg,pt} \overline{Int2}$.

In order to do this, we build an interface task-PIOA family $\overline{Ifc}$ that represents the common parts of $\overline{Int1}$ and $\overline{Int2}$. Then, we prove two claims.

1. $\overline{Int1} \leq_{neg,pt} \overline{SH} \| \overline{Ifc} \| \overline{Adv}$ and $\overline{SHR} \| \overline{Ifc} \| \overline{Adv} \leq_{neg,pt} \overline{Int2}$.
   We prove this by exhibiting simple, constant-bounded simulation relations between these systems.
2. $\overline{SH} \| \overline{Ifc} \| \overline{Adv} \leq_{neg,pt} \overline{SHR} \| \overline{Ifc} \| \overline{Adv}$. For this, we use our definition, Definition 31, of hard-core predicates, the fact that both $\overline{Ifc}$ and $\overline{Adv}$ are polynomial-time-bounded, and the composition property of $\leq_{neg,pt}$ (Theorem 5).

## 8 Conclusions

In this paper, we have described the *Time-Bounded Task-PIOA* framework, an automata-theoretic framework that is intended for modeling and analyzing security protocols. This framework extends the well-known Probabilistic I/O Automata framework [Seg95,SL95], by adding a new *task* mechanism for resolving nondeterministic choices and new ways of expressing bounds on computational resources. Time-bounded task-PIOAs have a simple theory, including composition and hiding operations and associated theorems, notions of perfect implementation ($\leq_0$) and approximate implementation ($\leq_{neg,pt}$), and probabilistic simulation relations that are sound for proving implementation relations.

We have described how our framework can be used to express some standard security notions, such as protocols, the adversaries with which they must contend, and the functional correctness and computational security properties they must satisfy. We have shown (via an example) how time-bounded task-PIOAs can be used to reformulate definitions of cryptographic primitives, and how such reformulated definitions can be used in proofs of computational security.

We have illustrated the use of our framework by modeling and analyzing functional correctness and computational security for an Oblivious Transfer protocol [EGL85,GMW87]. Our proofs are decomposed into a series of stages, each showing an implementation relation, $\leq_{neg,pt}$, between two systems. Most of these implementation relations are proved using simulation relations to match corresponding events and probabilities in the two systems. Others are proved using reduction arguments involving an underlying computational indistinguishability assumption, reformulated in terms of $\leq_{neg,pt}$. This paper is based on more detailed developments in [CCK+06c,CCK+06a,CCK+06b,CCK+05].

In current work, we are establishing general security protocol composition theorems, in the style of [Can01,PW01]. Following [PW01], we would also like to develop systematic ways of modeling common patterns of adversarial behavior and systematic ways of proving security properties of protocols that interact with these adversaries.

We believe that the model and techniques presented here will provide a good basis for analyzing a wide range of cryptographic protocols, especially those that depend on computational assumptions and achieve computational security. It remains to demonstrate this by applying the model and methods to many more case studies. We plan to apply our methods to more complex protocols, including protocols that use a combination of cryptographic primitives and protocols that

work against different types of adversaries. We would also like to apply the model and methods to study security protocols that have not yet been the subject of much formal study, such as *timing-based* and *long-lived* security protocols.

*Acknowledgments:* We thank Frits Vaandrager for help in the earlier stages of our project, and Silvio Micali for encouraging us to generalize our earlier definition of task-PIOAs to enhance the branching power of adversaries.

## References

[AR02] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[BCT04] G. Barthe, J. Cerderquist, and S. Tarento. A machine-checked formalization of the generic model and the random oracle model. In *Automated Reasoning: Second International Joint Conference (IJCAR)*, number 3097 in LNCS, pages 385–399, 2004.

[Bla05] B. Blanchet. A computationally sound mechanized prover for security protocols. Cryptology ePrint Archive, Report 2005/401, 2005. `http://eprint.iacr.org/`.

[Bla06] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *IEEE Symposium on Security and Privacy*, pages 140–154, Oakland, California, May 2006.

[BPW03] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, 2003.

[BPW04a] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *First Theory of Cryptography Conference (TCC 2004)*, volume 2951 of *LNCS*, pages 336–354, 2004.

[BPW04b] M. Backes, B. Pfitzmann, and M. Waidner. Secure asynchronous reactive systems. Cryptology ePrint Archive, Report 2004/082, 2004. `http://eprint.iacr.org/`.

[BR04] M. Bellare and P. Rogaway. The game-playing technique and its application to triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004. `http://eprint.iacr.org/`.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd Annual Conference on Foundations of Computer Science (FOCS)*, 2001. Full version available at `http://eprint.iacr.org/2000/067`.

[CCK+05] Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Using probabilistic i/o automata to analyze an oblivious transfer protocol. Cryptology ePrint Archive, Report 2005/452, 2005. `http://eprint.iacr.org/`. Version of February 16, 2007.

[CCK+06a] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Task-structured probabilistic I/O automata. In *Proceedings of the 8th International Workshop on Discrete Event Systems (WODES)*, Ann Arbor, Michigan, July 2006.

[CCK+06b] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Task-structured probabilistic I/O automata. Technical Report MIT-CSAIL-TR-2006-060, CSAIL, MIT, Cambridge, MA, 2006. Submitted for journal publication. Most current version available at `http://theory.csail.mit.edu/~lcheung/papers/task-PIOA-TR.pdf`.

[CCK+06c] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Using probabilistic I/O automata to analyze an oblivious transfer protocol. Technical Report MIT-CSAIL-TR-2006-046, CSAIL, MIT, 2006. This is the revised version of Technical Reports MIT-LCS-TR-1001a and MIT-LCS-TR-1001.

[CCLP07] Ran Canetti, Ling Cheung, Nancy Lynch, and Olivier Pereira. On the role of scheduling in simulation-based security. In *Proceedings of the 7th International Workshop on Issues in the Theory of Security (WITS'07)*, 2007. To appear.

[CH06] Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In Shai Halevi and Tal Rabin, editors, *Proceedings, Theory of Cryptography Conference (TCC)*, volume 3876 of *LNCS*, pages 380–403. Springer, March 2006. Full version available on `http://eprint.iacr.org/2004/334`.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing*, pages 494–503. AMCM, 2002.

[CM97] C. Cachin and U. M. Maurer. Unconditional security against memory-bounded adversaries. In Burt Kaliski, editor, *Advances in Cryptology - Crypto '97*, pages 292–306, Berlin, 1997. Springer-Verlag. Lecture Notes in Computer Science Volume 1294.

[DY83] D. Dolev and A.C. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29):198–208, 1983.

[EGL85] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *CACM*, 28(6):637–647, 1985.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the 19th Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[Gol01] Oded Goldreich. *Foundations of Cryptography Volume I Basic Tools*. Cambridge University Press, 2001.

[Gol04] Oded Goldreich. *Foundations of Cryptography, Volume II Basic Applications*. Cambridge University Press, 2004.

[Hal05] S. Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181, 2005. `http://eprint.iacr.org/`.

[Küs06] R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006)*, pages 309–320. IEEE Computer Society, 2006.

[LMMS98] P.D. Lincoln, J.C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the 5th ACM*

*Conference on Computer and Communications Security (CCS-5)*, pages 112–121, 1998.

[LSV] N. Lynch, R. Segala, and F. Vaandrager. Observing branching structure through probabilistic contexts. *Siam Journal on Computing*. To appear.

[MMS03] P. Mateus, J.C. Mitchell, and A. Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time calculus. In *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR)*, volume 2761 of *LNCS*, pages 327–349, 2003.

[MQU07] Jörn Müller-Quade and Dominique Unruh. Long-term security and universal composability. In *Theory of Cryptography, Proceedings of TCC 2007*, Lecture Notes in Computer Science. Springer-Verlag, March 2007. Preprint on IACR ePrint 2006/422, to be published.

[MW04] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proceedings of the First Theory of Cryptography Conference*, pages 133–151, Cambridge, MA, USA, 2004. Springer-Verlag - LNCS Vol. 2951.

[PSL00] A. Pogosyants, R. Segala, and N. Lynch. Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study. *Distributed Computing*, 13(3):155–186, 2000.

[PW00] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000.

[PW01] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–200, 2001.

[RMST04] A. Ramanathan, J.C. Mitchell, A. Scedrov, and V. Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. In *Proceedings of Foundations of Sotware Science and Computation Structires (FOSSACS)*, volume 2987 of *LNCS*, pages 468–483, 2004.

[Seg95] Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, May 1995. Also, MIT/LCS/TR-676.

[Sho04] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/`.

[SL95] Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, August 1995.

[SV99] M.I.A. Stoelinga and F.W. Vaandrager. Root contention in IEEE 1394. In *Proc. 5th International AMAST Workshop on Formal Methods for Real-Time and Probabilistic Systems*, volume 1601 of *LNCS*, pages 53–74. Springer-Verlag, 1999.