Reasoning About Probabilistic Security Using Task-PIOAs*

Aaron D. Jaggard**[1], Catherine Meadows***[2], Michael Mislove***[3], and Roberto Segala[4]

[1] Rutgers University, New Brunswick, NJ, adj@dimacs.rutgers.edu
[2] Naval Research Lab, Washington, DC, catherine.meadows@nrl.navy.mil
[3] Tulane University, New Orleans, LA, mwm@math.tulane.edu
[4] University of Verona, Italy, roberto.segala@univr.it

Abstract. Task-structured probabilistic input/output automata (task-PIOAs) are concurrent probabilistic automata that, among other things, have been used to provide a formal framework for the universal composability paradigms of protocol security. One of their advantages is that that they allow one to distinguish high-level nondeterminism that can affect the outcome of the protocol, from low-level choices, which can't. We present an alternative approach to analyzing the structure of task-PIOAs that relies on ordered sets. We focus on two of the components that are required to define and apply task-PIOAs: discrete probability theory and automata theory. We believe our development gives insight into the structure of task-PIOAs and how they can be utilized to model crypto-protocols. We illustrate our approach with an example from anonymity, an area that has not been previously been addressed using task-PIOAs. We model Chaum's Dining Cryptographers protocol at a level that does not require cryptographic primitives in the analysis. We show via this example, how our approach can leverage a proof of security in the case a principal behaves deterministically to prove security when that principal behaves probabilistically.

1 Introduction

Process algebras and other models of concurrent computation have become a staple for reasoning about security (cf. [1, 20], to name two examples). Originally, this work concentrated on the "Dolev-Yao" model in which cryptographic algorithms were modeled as black boxes. However, there is a growing body of research aimed at marrying concurrency theory with cryptographic reasoning. Prominent among these approaches is the work of Canetti, Lynch, Segala, et

* The authors wish to thank the Center for Discrete Mathematics and Theoretical Computer Science and the DIMACS Special Focus on Communication Security and Information Privacy for their support that allowed us to gather at Rutgers to work on this project for a week in May/June, 2008.
** The author wishes to acknowledge the support of the NSF and of the ONR during this research.
*** These authors wish to acknowledge the support of the ONR during this research.

al. [6], which uses *Task-PIOAs* to provide a formalization of Canetti's Universal Composability model [5]. Task-PIOAs are probabilistic input/output automata equipped with *tasks*, which define their execution sequences. The approach is highlighted by two aspects: First, this approach encodes the basic constituents of cryptography – one-way functions, hardcore predicates, etc., into the model and reasons about them in proving security properties.

The second distinguishing aspect of [6] is that it divides the nondeterminism in a concurrent system into high-level choices that can affect the outcome of the protocol, and low-level nondeterminism that does not affect the outcome. By allowing the adversary to see the outcomes only of the low-level choices, the model overcomes the problem that concurrency models generally afford the adversary too much power by allowing it to see the outcomes of all the choices made during the execution of a protocol.

Because of all this, the task-PIOA work is often cited as a promising approach to modeling crypto-protocols, but at the same time this makes the task-PIOA analysis of oblivious transfer [6], for example, rather daunting. Indeed, there are three components that must be mastered to appreciate the subtleties of task-PIOAs: the theory of discrete probability that is applied to task-PIOAs, the theory of probabilistic automata that underlies task-PIOAs, and the incorporation of cryptographic primitives that requires detailed and often-tedious reasoning. We note that this rather formidable array of components is inherent in all approaches to marrying formal with cryptographic reasoning, and is not specific to task-PIOAs. Thus, all approaches to marrying formal with cryptographic reasoning suffer from the same complexity. Our approach is to separate the cryptographic primitives in the model from the other aspects, and to treat these other facets first, reserving a similar analysis of the cryptographic components for future work.

Thus, the purpose of the current paper is to present the first two components – discrete probability theory applied to task-PIOAs, and the theory of probabilistic automata needed to analyze task-PIOAs, from a different perspective, one we hope will make the technology more easily digested. Our approach relies on ordered sets as an underlying theme for both discrete probability and for the automata theory needed to describe task-PIOAs.

The universal composability approach to proving security employs the notion of simulator as part of a mechanism that allows comparison of "real-world" processes with "ideal processes" that are secure by design. Task-PIOAs use simulation relations common to cryptographic proofs to carry out such comparisons. Simulation relations validate security properties by showing the trace distributions on visible events of the real-world process are contained in those of the simulating process. As a result, an unbiased observer cannot tell if it is interacting with the real-world process or with the simulator, and the security of the real-world process follows.

We illustrate our results with an example in which cryptographic primitives don't play a role. This allows us to demonstrate the techniques developed in this paper to model a security protocol, avoiding cryptographic details that are

independent of our approach to combining discrete probability theory with the theory of probabilistic automata. The example comes from anonymity: it is the canonical example of Chaum's Dining Cryptographers protocol [9]. Our approach also has the advantage that it explicitly models a method by which the choice of who pays is made, a detail left unspecified in Chaum's original paper, and one that is also left unspecified in other approaches such as [4].

Task-PIOAs use task schedules to resolve the nondeterminism, and a task-PIOA simulation relation typically shows that each task schedule for a real-world process has a corresponding schedule for its simulating process. In this paper we introduce a somewhat simpler notion of simulation we call *Task-PIOA maps*. The main difference is that, while task-PIOA simulations utilize maps between task schedules, Task-PIOA maps map states to states, actions to actions, and tasks to tasks. This makes defining a task-PIOA map somewhat simpler, and their use is well-suited for indistinguishability proofs in which the only difference between the two processes is the choice of a secret. This is the case for our proof of anonymity of Dining Cryptographers. Anticipating some details, the protocol is secure if the probability distributions on the announcements of the cryptographers doesn't vary depending on which one is paying. Our analysis shows that the distributions in the case of a probabilistic choice of who pays can be calculated compositionally from the distributions that arise from its deterministic subprocesses. Our argument uses ideas from [4, 7] relying on conditional probability.

The rest of the paper is organized as follows. In the next section, we give a comparison with other work. The following sections give some results about discrete probability theory based on order-theoretic arguments, and the basic definitions and results used by task-PIOAs. Following that, we give the main results of the paper which define Task-PIOA maps and show they preserve trace distributions of task-based probabilistic input/output automata. This is followed by a brief description of the Dining Cryptographers protocol and how to encode it using process algebra. We next encode the Dining Cryptographers Protocol using a task-based probabilistic input/output automata and then show that the protocol is secure if the choice of who pays is either deterministic or probabilistic. In the final section we summarize our results and discuss possible future work.

2 Comparison with other work

As we already commented, the use of process algebras and of models of concurrency is now commonplace in analyzing crypto-protocols, as well as other areas of security. What is novel about the task-PIOA approach as presented in [6] is the level of detail at which processes are treated. Most security models operate under the traditional Dolev-Yao assumption and assume perfect encryption. However, these assumptions have been shown to be unrealistic in some contexts [17], which is one of the impetuses for the universal composability approach [5] to modeling security. This need for more realism has been accompanied by a myriad of details that make analyses much more intricate. In particular, in what is probably the most closely-related work to the task-PIOA approach, [17, 20], Probabilistic Polynomial-time Turing Machines are used as the basis for a model. This

approach also involves myriad details about computations, details we seek to abstract away in our approach to understanding parts of the PIOA approach.

Part of the problem with these more realistic models is the level of detail, but an added problem with [6] is the presentation of the model. Indeed, the task-PIOA approach is composed of two parts: (i) the presentation of task-based probabilistic input/output automata (task-PIOAs), and (ii) the application of these automata to model cryptographic primitives in the analysis of the oblivious transfer protocol. Here, we give an alternative presentation of task-PIOAs, using techniques from order theory. While our approach requires familiarity with ordered sets, the mathematical results needed are not especially deep, and the ordered sets approach carries with it computational intuitions that are not so clear in [6].

The Dining Cryptographers protocol originated in the work of Chaum [9], and it quickly became a prototypical test example for modeling anonymity. For example, Schneider and Sidiropoulos [21] use CSP to analyze the protocol, after first replacing probabilistic choice by nondeterministic choice. While CSP is limited to nondeterministic choice, there are models of concurrency that support both nondeterminism and probabilistic choice, and these require a mechanism to account for how the nondeterminism in the system is resolved. The most commonly used approach is to include a *scheduler* that resolves the nondeterministic choices. In security analyses, the scheduler is often seen as an adversary that can control the occurrence of events in order to maximize the chances of compromising the security of the system. Garcia et al. [11] introduce *admissible schedulers* that limit the power of the adversary so that it is "realistic" and cannot see the outcomes of choices that should be hidden from it.

Probably the most exhaustive treatment of the Dining Cryptographers using concurrency appears in the papers [4,7,8]. In [4], an analysis shows the limitations of substituting nondeterminism for probabilistic choice, as was done in [21], and analyses are presented of the Dining Cryptographers with users that are nondeterministic or probabilistic in their choices, under some added hypotheses. In [7], the pi-calculus is augmented with labels guarding choices, so that the scheduler can enable a choice, but not control or see its outcome. In [8], a new notion of *conditional capacity* of a channel is introduced, and this is used to analyse the degree of anonymity of a system in which some information is leaked on purpose. Finally, [14] applies simulation relations based on coalgebras to analyze the Dining Cryptographers. As is typical of other treatments, the presentation is at an abstract level, concentrating solely on the case in which the choice of who pays is deterministic, and in which the implementation of the choice is left unspecified. Moreover, this model does not support nondeterminism, and presents everything in a purely probabilistic framework.

There is a growing body of literature on modeling probabilistic processes, including models of concurrency that support both nondeterminism and probabilistic choice (see, e.g., [18]), as is the case here. A major theme along this line stems from the seminal work of Larsen and Skou [16] on labeled Markov processes (see also [10, 19]). A major difference between task-PIOAs and that

work is the nature of the transition relation that defines the evolution of the processes. We comment in more detail on this point in Section 9. Nevertheless, it should be noted that the Task-PIOA maps we introduce are a form of functional simulation.

In summary, our main results are as follows:

- The use of order theory to analyze task-PIOAs, and in particular, the Apply operator that applies a task to a probabilistic input/output automaton, which make clear how processes evolve under the application of task schedules.
- The notion of a *Task-PIOA map* and the proof that these maps take trace distributions of a domain task-PIOA to trace distributions of the target task-PIOA. These mappings show how one task-PIOA has its trace distributions observable events contained in those of another using the same task schedule.
- An analysis of the Dining Cryptographers Protocol using task-PIOAs. Our version of Dining Cryptographers uses a Master that chooses who pays (either one of the cryptographers or the Master itself), a concept introduced in [4]. We prove the anonymity of the protocol from the point of view of the cryptographers when the Master is deterministic, and then use that result to show that anonymity also holds in the case of a probabilistic Master.

3 Ordered Structures and Domain-theoretic Arguments

In this section we present the results on ordered sets and their application to models of computation we need to analyze probabilistic input/output automata. We emphasize at the outset that our results are valid in the case of directed complete partial orders (e.g., *domains*) but we restrict the presentation to the case of ordered sets, since that's all that is needed in the current setting.

Definition 1. *An* ordered set *is a non-empty set together with a partial order, i.e., a reflexive, antisymmetric and transitive relation. If P and Q are ordered sets, and if $f\colon P \to Q$ is a mapping between them, then*

- *f is* monotone *if $x \leq_P y$ implies $f(x) \leq_Q f(y)$.*
- *f is* progressive *if $P = Q$ and $x \leq_P f(x)$ for all $x \in P$.*

We use Ord *to denote the category of ordered sets and monotone maps.*

The prototypical example of an ordered set is the family A^* of finite words over an alphabet A in the *prefix order:* $s \leq t \;\Leftrightarrow\; (\exists u \in A^*)\; t = su$.

If X is a set, then a *discrete probability measure over* X has the form $\sum_{i\in I} r_i \delta_{x_i}$, where I is a countable set, δ_{x_i} denotes the point mass at $x_i \in X$ and $\sum_i r_i = 1$, where $0 < r_i$ for each i. If I is finite, then the measure is said to be *simple*. For $\mu = \sum_{i\in I} r_i \delta_{x_i}$, then the *support of* μ is defined by $\mathsf{supp}\,\mu = \{x_i \mid i \in I\}$.

If P is an ordered set, the order on P lifts to the family $\mathsf{Disc}(P)$ of discrete probability measures on P: $\sum_i r_i \delta_{x_i} \leq \sum_j s_j \delta_{y_j}$ iff $\sum\{r_i \mid x_i \in A\} \leq \sum\{s_j \mid y_j \in A\}$ for each *upper set* A.[5] Thus, $\mu \leq \nu$ iff $\mu(A) \leq \nu(A)$ for every upper set A. For simple measures $\sum_{i=1}^m r_i \delta_{x_i} \leq \sum_{j=1}^n s_j \delta_{y_j}$ iff there is a family $t_{ij} \geq 0$,

[5] A is an *upper set* if $A = \uparrow A = \{x \in P \mid (\exists a \in A)\; a \leq_P x\}$.

$1 \leq i \leq m$ and $1 \leq j \leq n$ satisfying:

(i) $(\forall i)\ r_i = \sum_j t_{ij}$, (ii) $(\forall j)\ \sum_i t_{ij} = s_j$, and (iii) $t_{ij} > 0 \ \Rightarrow\ x_i \leq_P y_j$.

(Cf. Theorem 4.10 of [15] for details.) There is a monad $\mathsf{Disc}\colon \mathsf{Ord} \to \mathsf{Ord}$ that associates to each ordered set P the family $\mathsf{Disc}(P)$ of discrete probability measures on P and that sends each monotone mapping $f\colon P \to Q$ to the mapping $\mathsf{Disc}(f)\colon \mathsf{Disc}(P) \to \mathsf{Disc}(Q)$ by $\mathsf{Disc}(f)(\sum_i r_i\delta_{x_i}) = \sum_i r_i\delta_{f(x_i)}$. The unit of the monad is the mapping $x \mapsto \delta_x$ and the multiplication $m\colon \mathsf{Disc}(\mathsf{Disc}(P)) \to \mathsf{Disc}(P)$ is just integration: $\sum_i r_i\delta_{\sum_j s_{ij}\delta_{x_{ij}}} = \sum_{ij} r_i s_{ij}\delta_{x_{ij}}$.

4 Probabilistic Input/Output Automata

We use task-structured probabilistic input/output automata as our basic model of computation. This notion is due to Canetti, Lynch, et. al. [6]. In this section, we summarize the main ideas from [6] that are necessary to understand this paper.

Definition 2. *A* Probabilistic Input/Output Automaton $\mathcal{A} = (Q, \overline{q}, I, O, H, D)$ *is a* 6*-tuple where:*

1. *Q is a countable set of* states *containing $\overline{q} \in Q$ as the* start state*;*
2. *I, O and H are pairwise disjoint, countable sets of actions, referred to as* input, output *and* internal actions*, respectively. The set $\mathsf{Act} ::= I \cup O \cup H$ is called the set of* actions *of $\mathcal{A}$. The set of* external *actions of $\mathcal{A}$ is $E ::= I \cup O$ and the set of* locally controlled *actions is $L ::= O \cup H$.*
3. *$D \subseteq Q \times \mathsf{Act} \times \mathsf{Disc}(Q)$ is a* transition relation*. An action a is* enabled *in state q if $(q, a, \mu) \in D$ for some $\mu \in \mathsf{Disc}(Q)$, which we denote by $\mu_{q,a}$.*
   We also assume D satisfies:
   Input enabling: *For every $q \in Q$ and $a \in I$, a is enabled in q.*
   Transition determinism: *For every $q \in Q$ and $a \in \mathsf{Act}$, there is at most one $\mu \in \mathsf{Disc}(Q)$ such that $(q, a, \mu) \in D$*

Probabilistic input/output automata also admit a hiding operator*: If $S \subseteq O_{\mathcal{A}}$ is a set of output actions of $\mathcal{A}$, then $\mathcal{A} \setminus S$ denotes the process where $O = O_{\mathcal{A}} \setminus S$ and $H = H_{\mathcal{A}} \cup S$. The result is that the output of actions in S are visible only to the process executing them, and to the recipient processes of the actions.*

A probabilistic input/output automaton can be viewed as a labelled transition system where the transition relation is a partial function $\Delta\colon Q \times \mathsf{Act} \rightharpoonup \mathsf{Disc}(Q)$ defined by $\Delta(q, a) = \mu_{q,a}$, if such a measure exists. Input enabling implies the restriction $\Delta|_{Q\times I}\colon Q \times I \to \mathsf{Disc}(Q)$ is a total function.

We also need a method for composing PIOAs.

Definition 3. *Let $\mathcal{A}_i = (Q_i, \overline{q}_i, I_i, O_i, H_i, D_i)$, $i = 1, 2$ be PIOAs. We say $\mathcal{A}_1$ and $\mathcal{A}_2$ are* compatible *if $\mathsf{Act}_1 \cap H_2 = \mathsf{Act}_2 \cap H_1 = \emptyset = O_1 \cap O_2$. If this is the case, then we define $\mathcal{A}_1 \| \mathcal{A}_2 = (Q_{\mathcal{A}_1\|\mathcal{A}_2}, \overline{q}_{\mathcal{A}_1\|\mathcal{A}_2}, I_{\mathcal{A}_1\|\mathcal{A}_2}, O_{\mathcal{A}_1\|\mathcal{A}_2}, H_{\mathcal{A}_|\|\mathcal{A}_2}, D_{\mathcal{A}_1\|\mathcal{A}_2})$ where:*

- $Q_{\mathcal{A}_1\|\mathcal{A}_2} = Q_1 \times Q_2$, $\overline{q}_{\mathcal{A}_1\|\mathcal{A}_2} = \langle \overline{q}_1, \overline{q}_2 \rangle$,
- $I_{\mathcal{A}_1\|\mathcal{A}_2} = (I_1 \cup I_2) \setminus (O_1 \cup O_2)$, $O_{\mathcal{A}_1\|\mathcal{A}_2} = O_1 \cup O_2$, *and* $H_{\mathcal{A}_1\|\mathcal{A}_2} = H_1 \cup H_2$,

– $D_{\mathcal{A}_1 \| \mathcal{A}_2} = \{(\langle q_1, q_2 \rangle, a, \mu_1 \times \mu_2) \mid (a \in \mathsf{Act}_i \;\Rightarrow\; (q_i, a, \mu_i) \in D_i),$
and $(a \notin \mathsf{Act}_j \;\Rightarrow\; \mu_j = \delta_{q_j})\}$.

The main points to note of the composition are that (1) the inputs of either component can be provided by outputs of the other component, so the composition of processes can be *closed* – its set of inputs can be empty, and (2) if an action is in the alphabet of only one component, then the other component "idles" in its current state while the enabled component executes the action. However, even though one component may be able to perform an action in a given state, the other component can block the action if the action is in its alphabet, but it cannot execute the action in its current state.

4.1 Tasks

Tasks provide a mechanism for resolving nondeterminism – the choice of which action to execute in a given state, in the execution of a PIOA.

Definition 4. *A* Task Probabilistic Input/Output Automaton *is a pair* $\mathcal{T} = (\mathcal{A}, \mathcal{R})$ *where*

– $\mathcal{A} = (Q, \overline{q}, I, O, H, D)$ *is a probabilistic I/O automaton, and*
– $\mathcal{R} \subseteq (O \times O) \cup (H \times H)$ *is an equivalence relation on the locally controlled actions. The equivalence classes of* $\mathcal{R}$ *are called* tasks.

A task-PIOA $\mathcal{T}$ *is said to be* action deterministic *if for each state* $q \in \mathcal{A}$ *and each task* $T \in \mathcal{R}$*, there is at most one action* $a \in T$ *that is enabled in* q.

A task schedule *for a task-PIOA* $\mathcal{T}$ *is a finite sequence* $\rho = T_1 T_2 \cdots T_n$ *of tasks in* $\mathcal{R}$.[6]

4.2 Applying Task Schedules

Since we have no final states in Q, there is a question of what constitutes a computation of a task-PIOA. We use task schedules to specify the sequences of actions a process should execute. The probability distribution generated by the process as it executes that sequence of tasks then defines a computational behavior of the process.

Our interest is in the trace distributions over finite sequences of observable events of the automaton $\mathcal{A}$. For an automaton $\mathcal{A} = (Q, \overline{q}, I, O, H, D)$, we let

$$\mathsf{Exec}^*\mathcal{A} = \{\overline{q} a_0 \ldots q_{n-1} a_{n-1} q_n \mid q_i \in Q, a_i \in \mathsf{Act}, q_{i+1} \in \mathsf{supp}\, \mu_{q_i, a_i}\ (\forall i < n)\}.^{7}$$

Note that $\mathsf{Exec}^*\mathcal{A}$ is ordered by the prefix ordering as a subset of $Q \times (\mathsf{Act} \times Q)^*$. If $a \in \mathsf{Act}$, then we define the set of finite execution fragments whose last state enables the action a:

$$A_a = \{\alpha \in \mathsf{Exec}^*\mathcal{A} \mid a \text{ enabled in } \mathsf{lstate}(\alpha)\},$$

[6] Our results in general include a treatment of the case of infinite task schedules (in which case the utility of domain theory comes to the fore), but since we only need finite task schedules in the present setting, we limit our presentation to that case. Nonetheless, the order-theoretic approach pays a dividend by giving a clear and concise definition of the application of a task to a process.

[7] Note that all such sequences begin at $\overline{q}$. Recall $x \in \mathsf{supp}\, \mu$ iff $\mu(x) > 0$.

where $\mathsf{lstate}(\alpha) = q_n$ is the final state of $\alpha = \overline{q}a_0 \cdots a_{n-1}q_n$. Likewise, for a task T, let A_T be the set of finite execution fragments whose last state enables the task T:

$$A_T = \{\alpha \in \mathsf{Exec}^*\mathcal{A} \mid (\exists a \in T)\ a \text{ enabled in } \mathsf{lstate}(\alpha)\}.$$

For each finite execution fragment $\alpha \in A_T$, if the action $a \in T$ is enabled in $\mathsf{lstate}(\alpha)$, we let $\mu_{\mathsf{lstate}(\alpha),a}$ be the target measure of the unique transition $(\mathsf{lstate}(\alpha), a, \mu_{\mathsf{lstate}(\alpha),a}) \in D$. Then we define

$$\mathsf{Apply}(\mu, T) = \sum_{\alpha \notin A_T} \mu(\alpha)\delta_\alpha + \sum_{\alpha \in A_T} \mu(\alpha) \left(\sum_s \mu_{\mathsf{lstate}(\alpha),a_T}(s)\delta_{\alpha a_T s} \right) \text{[8]} \qquad (1)$$

Note that execution sequences in whose last state T is not enabled still appear as the first summand on the right in the definition of $\mathsf{Apply}(\mu, T)$; this is why we cannot use the traditional approach of adjoining a deadlock state to transform the transition relation $\Delta\colon Q \times \mathsf{Act} \rightharpoonup \mathsf{Disc}(Q)$ into a total function.

Thus, Apply applies the task T to each execution fragment $\alpha \in \mathsf{supp}\,\mu$, resolving the nondeterminism of which action should occur next. For a finite task schedule $\rho = T_1 \cdots T_n$, we define

$$\mathsf{Apply}(\mu, \rho) = \mathsf{Apply}(\mathsf{Apply}(\mu, T_1 \cdots T_{n-1}), T_n).$$

Proposition 1. *Let $\mu = \sum_\alpha \mu(\alpha)\delta_\alpha \in \mathit{Disc}(\mathit{Exec}^*(\mathcal{A}))$, then*

$$\mu \leq \mathit{Apply}(\sum_\alpha \mu(\alpha)\delta_\alpha, \rho) = \sum_\alpha \mu(\alpha)\mathit{Apply}(\delta_\alpha, \rho).$$

In particular, $\mathit{Apply}(\mu, \rho)$ is well-defined for any task sequence $\rho = T_1 \cdots T_n$.

Proof. By induction on the lengh of ρ – see Appendix.

Corollary 1. *If $\mu = \sum_i p_i\mu_i$, then $\mu \leq \mathit{Apply}(\sum_i p_i\mu_i, \rho) = \sum_i p_i\mathit{Apply}(\mu_i, \rho)$.*

Proof. By the Proposition, we know that $\mu_i \leq \mathsf{Apply}(\mu_i, \rho)$ for each index i, so

$$\mu = \sum_i p_i\mu_i \leq \sum_i p_i\mathsf{Apply}(\mu_i, \rho) = \mathsf{Apply}(\sum_i p_i\mu_i, \rho),$$

the last equality following by expanding each μ_i as a convex sum of point masses and then applying the Proposition once again.

5 Comparing PIOAs

An important component of any modeling paradigm is the ability to compare models of distinct processes, in order to analyze the differences between them. In the development of task-PIOAs in [6], this is done via *simulation relations*, which are used to show that the probability distributions over observable behaviors of one task-PIOA are a subset of those of another task-PIOA. This is

[8] Since each task T allows only one action $a_T \in T$ to be enabled in $\mathsf{lstate}(\alpha)$, there is no sum over actions in the second summand on the right.

useful for simplifying a process by devising a simpler, *simulating process*, and then deductions about the behavior of the simpler process can be used to prove properties about the more complicated one. This approach is based on matching a task schedule for the first process to one for the simulating process, but in our setting, the security specification requires we use *the same task schedule* for both processes. Rather than use the task-PIOA simulation relations from [6], we present a simpler approach to simulation that meets this requirement. The setting is as follows:

If $\mathcal{T} = (\mathcal{A}, \mathcal{R})$ is a task-PIOA, and $A \subseteq (I \cup O)$ is a set of *observable actions*, then $\mathsf{trace}_A \colon \mathsf{Exec}^*\mathcal{A} \to A^*$ is the function that extracts the sequence of A-actions (i.e., the sequences from A^*) from an execution sequence of $\mathcal{A}$. Since we are using only discrete measures, all sets are measurable, and so trace_A is a measurable map. Then $\mathsf{Disc}(\mathsf{trace}_A)(\mu) = \mu \circ \mathsf{trace}_A^{-1}$ is a measure on A^* for each measure $\mu \in \mathsf{Disc}(\mathsf{Exec}^*\mathcal{A})$. In fact, if $\mu = \sum_i r_i \delta_{\alpha_i}$, then $\mathsf{Disc}(\mathsf{trace}_A)(\mu) = \sum_i r_i \delta_{\mathsf{trace}_A(\alpha_i)}$. We let $\mathcal{S}_\mathcal{A}$ denote the family of task schedules on $\mathcal{A}$, and for $\rho \in \mathcal{S}_\mathcal{A}$, let $\mathsf{Apply}(\delta_{\overline{q}}, \rho) = \sum_{\alpha \in \mathsf{supp}\,\mathsf{Apply}(\delta_{\overline{q}}, \rho)} r_\alpha \delta_\alpha$. Then we define

$$\mathsf{tdist}_A^\rho(\mathcal{A}) = \mathsf{Disc}(\mathsf{trace}_A)(\mathsf{Apply}(\delta_{\overline{q}}, \rho)) = \sum_{\alpha \in \mathsf{supp}\,\mathsf{Apply}(\delta_{\overline{q}}, \rho)} r_\alpha \delta_{\mathsf{trace}_A(\alpha)}$$

be the measure on A^* induced by trace_A from the measure $\mathsf{Apply}(\delta_{\overline{q}}, \rho)$. We also call

$$\mathsf{tdists}_A(\mathcal{A}) = \{\mathsf{tdist}_A^\rho(\mathcal{A}) \mid \rho \in \mathcal{S}_\mathcal{A}\}$$

the *A-trace distributions* of $\mathcal{A}$. We elide the subscript if $A = I \cup O$.

Definition 5. *Let $\mathcal{A}_i = (Q_i, \overline{q}_i, I_i, O_i, H_i, D_i), i = 1, 2$ be two compatible task-PIOAs, with task sets $\mathcal{T}_i, i = 1, 2$, respectively. We define a* Task-PIOA map *$\phi \colon \mathcal{A}_1 \to \mathcal{A}_2$ to consist of*

- *A mapping $\phi \colon Q_1 \to Q_2$ of the states of $\mathcal{A}_1$ to the states of $\mathcal{A}_2$ satisfying $\phi(\overline{q}_1) = \overline{q}_2$. We call this the* state component *of ϕ.*
- *A mapping $\phi \colon \mathsf{Act}_1 \to \mathsf{Act}_2$ of actions of $\mathcal{A}_1$ to actions of $\mathcal{A}_2$. We call this the* action component *of ϕ.*

We also require the mapping $\phi \colon \mathcal{A}_1 \to \mathcal{A}_2$ to satisfy the compatibility conditions:

(i) $\langle q, a, \mu \rangle \in D_1 \;\Rightarrow\; \langle \phi(a), \phi(a), \mathsf{Disc}(\phi)(\mu) \rangle \in D_2$*; i.e., if $q \in Q_1$ and $a \in \mathsf{Act}_1$ is enabled in state q, then $\phi(a)$ is enabled in state $\phi(q)$ and $\mathsf{Disc}(\phi)(\mu_{q,a}) = \mu_{\phi(q),\phi(a)}$.*

(ii) For each $T \in \mathcal{T}_1$, $\phi(T) \in \mathcal{T}_2$.

Remark 1. Note that if ϕ does not rename states or actions, then this coincides with the notion of *refinement* in process algebra.

The map ϕ induces $\mathsf{Exec}^*\phi \colon \mathsf{Exec}^*\mathcal{A}_1 \to \mathsf{Exec}^*\mathcal{A}_2$ by $\mathsf{Exec}^*\phi(\overline{q}a_0 \cdots a_{n-1}q_n) = \phi(\overline{q})\phi(a_0) \cdots \phi(a_{n-1})\phi(q_n)$, which in turn gives $\mathsf{Disc}(\mathsf{Exec}^*\phi) \colon \mathsf{Disc}(\mathsf{Exec}^*\mathcal{A}_1) \to \mathsf{Disc}(\mathsf{Exec}^*\mathcal{A}_2)$ by $\mathsf{Disc}(\mathsf{Exec}^*\phi)(\sum_i r_i \delta_{\alpha_i}) = \sum_i r_i \delta_{\mathsf{Exec}^*\phi(\alpha_i)}$. Our main result is:

Theorem 1. *Let $\phi\colon \mathcal{A}_1 \to \mathcal{A}_2$ be a Task-PIOA map of compatible task-PIOAs satisfying $\phi(A_T) = A_{\phi(T)}$ for each task $T \in \mathcal{T}_1$. If $A \subseteq \mathsf{Act}_1$ is the set of observable actions, then $\mathsf{Disc}(\phi|_A^*)(\mathsf{tdist}(\mathcal{A}_1)) \subseteq \mathsf{tdist}(\mathcal{A}_2)$.*

Proof. We give an outline of the proof here, and provide full details in the Appendix. First, if $\mu \in \mathsf{Disc}(\mathsf{Exec}^*\mathcal{A}_1)$, then $\mu = \sum_{\alpha \in \mathsf{supp}\,\mu} r_\alpha \delta_\alpha$. So, for $T \in \mathcal{T}_1$, we can apply our definition of Apply from Equation 1 and of $\mathsf{Disc}(\mathsf{Exec}^*\phi)$ above to show $\mathsf{Disc}(\mathsf{Exec}^*\phi)(\mathsf{Apply}(\mu, T)) = \mathsf{Apply}(\mathsf{Disc}(\mathsf{Exec}^*\phi)(\mu), \phi(T))$. An induction on n then shows the analogous result holds with any task schedule $\rho = T_1 \cdots T_n$ in place of T. The next step is to use this equation to show

$$\mathsf{Disc}(\phi|_A^*)(\mathsf{Disc}(\mathsf{trace})(\mathsf{Apply}(\mu, \rho))) = (\mathsf{Disc}(\mathsf{trace}) \circ \mathsf{Disc}(\mathsf{Exec}^*\phi))(\mathsf{Apply}(\mu, \rho)),$$

which implies the result.

6 Dining Cryptographers

The Dining Cryptographers Protocol is originally due to Chaum [9]. It postulates several cryptographers who are dining together. They are trying to determine whether one of them paid for dinner or NSA paid, without revealing to each other which cryptographer paid. In [9] the implementation of the decision of who pays is left unspecified; we use the solution of Bhargava and Palamadessi [4] in which a Master (that is, NSA) chooses who pays. The cryptographers use the following protocol to accomplish this. Each cryptographer has a coin that he flips, and he reports the outcome of that flip to the cryptographer to his right. The cryptographers then report the outcomes of the two coin flips they know – theirs and the one to their left – by announcing either *Agree* or *Disagree*. A cryptographer reports the outcome correctly if she is not paying, but if she is paying for dinner, then she reverses the announcement and lies about the outcome of the coin tosses she has witnessed. A simple argument based on parity shows that the Master is paying if the number of *Disagree* announcements is even, while one of the cryptographers is paying if there are an odd number of *Disagree* announcements. Moreover, if the cryptographers are honest and if the coins they use are fair, this protocol protects the identity of the paying cryptographer from the other cryptographers, as long as there are at least three cryptographers.

We will use probabilistic input/output automata [6] to model the Dining Cryptographers. The formal model will be introduced below, but first we give a process algebraic description of the component processes. We focus on the least number of cryptographers that makes the analysis meaningful, which is three. The processes are:[9]

$$\begin{aligned} \mathit{Master} ::= {} & \mathit{choose\_payer} \circ \mathit{pay}_0 \circ \mathit{pay}_1 \circ \mathit{pay}_2 \\ \mathit{Crypto}_i ::= {} & \mathit{rec\_pay}_i \circ \mathit{flip\_coin}_i \circ \mathit{learn\_coin}_{i-1} \circ \mathit{tell\_coin}_i \circ \\ & \mathit{compare\_coins} \circ \mathit{announce}_i \circ \mathit{receive\_announcements} \end{aligned}$$

The composition operator $\circ$ in this syntax is a meant to allow the actions to occur in any order specified by the environment. The action pay_i sends the message to

[9] We us indices $i = 0, 1, 2$ modulo 3, so $0 - 1 = 2$, $2 + 1 = 0$, etc.

cryptographer i, $i = 0, 1, 2$ indicating whether or not she is paying. We believe the meaning of the components of the processes listed here is clear, so they require no further explanation. But these names will change when we encode the processes as probabilistic input/output automata, in order to conform to the requirements of that system. The process representing our protocol is then the composition

$$Master \parallel Crypto_0 \parallel Crypto_1 \parallel Crypto_2,$$

where $\parallel$ denotes parallel composition.

7 A Model of Dining Cryptographers

We now show how to model the Dining Cryptographers using task-based probabilistic input/output automata. Figure 1 describes the Master process and its transition relation. This includes the set of actions for the process, labeled according to whether they are input, output or internal actions; it also gives the state space of the process and the list of tasks. The Master has an internally defined distribution $r \cdot \delta_{\langle master, \bot_{Q_0} \rangle} + \sum_i r_i \delta_{\langle crypto_i, \bot_{Q_0} \rangle}$ that gives the probability of choosing the payer – i.e., $r + \sum_i r_i = 1$. Also, the transition relation encodes the Master's actions of sending messages to each of the cryptographers indicating which one, if any, is paying, and the state space records the fact that the message to each cryptographer has been sent. The Master also observes the announcements of the cryptographers. Actions determine tasks:, e.g., one of the Master's tasks is $pay_i = \{pay_i(T), pay_i(F)\}$.

Master

Actions:

Input:

Output:
$pay_j(c), c \in \{T, F\}, j \in \{0, 1, 2\}$

Internal:
choose_payer

State:
$Payer \times Q_0$, where $Q_0 = \prod_{i=0}^{2} \{\bot, T\}$,
$Payer = \{\bot, Master, Crypto_i \mid i = 0, 1, 2\}$,
all initially $\bot$

Tasks:
$\{choose\_payer\}$,
$\{pay_j(c) \mid c \in \{T, F\}, j = 0, 1, 2\}$

Transitions:

$$D = \{\langle \langle \bot, \bot_{Q_0} \rangle, choose\_payer, r \cdot \delta_{\langle Master, \bot_{Q_0} \rangle} + \textstyle\sum_i r_i \delta_{\langle Crypto_i, \bot_{Q_0} \rangle} \rangle\}$$
$$\cup \{\langle \langle Master, q \rangle, pay_j(F), \delta_{\langle Master, q' \rangle} \mid q_j = \bot, q'_j = T,\ j = 0, 1, 2\}$$
$$\cup \{\langle \langle Crypto_i, q \rangle, pay_i(T), \delta_{\langle Crypto_i, q' \rangle} \rangle \mid q_i = \bot \ \&\ q'_i = T\}$$
$$\cup \{\langle \langle Crypto_i, q \rangle, pay_j(F), \delta_{\langle Crypto_j, q' \rangle} \rangle \mid q_j = \bot \ \&\ q'_j = T,\ j \neq i\}$$

Fig. 1: The Master automaton chooses the payer

Figure 2 gives the actions, state set and tasks of the cryptographers, while the transition relation is given in Figure 3. Together, these model the cryptographers as task-based probabilistic input/output automata. These are the most complicated task-PIOAs, having several states and actions to perform. In essence, each cryptographer receives a message from the Master indicating whether she is paying, and then uses this information in determining whether to honestly announce the outcome of the coin flips seen, or to lie. Each cryptographer also has an unbiased coin: this is specified by the output of $flip_i$ that is $\frac{1}{2}\delta_H + \frac{1}{2}\delta_T$. $Crypto_i$

reports the outcome of her coin flip to $Crypto_{i+1}$, compares her flip with the one from $Crypto_{i-1}$ and announces the outcome as described.[10] The component names have changed from the syntax in Section 5; for example, rec_pay_i is now the input action pay_i for $Crypto_i$, while $learn\_coin_{i-1}$ is shortened to the input action $coin_{i-1}$, etc.

$Crypto_i$
Actions:
Input:
$pay_i(c), c \in \{T, F\}$
$coin_{i-1}(c), c \in \{H, T\}$
(We again use subtraction mod 3.)
$announce_{i-1}(c), c \in \{Agree, Disagree\}$
$announce_{i+1}(c), c \in \{Agree, Disagree\}$

Output:
$coin_i(c), c \in \{H, T\}$
$announce_i(c), c \in \{Agree, Disagree\}$
Internal:
$flip_i$
$compare_i$

Tasks:
$\{flip_i\}, \{coin_i(c)\}, \{compare_i\}$,
$\{announce_i(c)\}$

State:
$Q_i = Pay_i \times Coin_{i-1} \times Coin_i \times Coin\_sent_i$
$\times Compare_i \times Announce_{i-1} \times Announce_{i+1}$
$Pay_i = \{pay_i(c) \mid c \in \{\perp, T, F\}\}$
initially $\perp$
$Coin_i = \{coin_{i-1}(c) \mid c \in \{\perp, H, T\}\}$
initially $\perp$
$Coin\_sent_i = \{coin\_sent_i(x) \mid x \in \{F, T\}\}$
initially F
$Compare_i = \{\perp, T, F\}$, initially $\perp$
$Announce_j = \{announce_j(c)\}, j = i-1, i+1$
$c \in \{\perp, Agree, Disagree\}$, initially $\perp$

Fig. 2: The actions and state set of $Crypto_i$

$Cryptographer_i$
Transitions:
$Q_i = Pay_i \times Coin_{i-1} \times Coin_i \times Coin\_sent_i \times Compare_i \times Announce_i$
$Pay_i = Compare_i = \{\perp, T, F\}$; $Coin_{i-1} = Coin_i = \{\perp, H, T\}$;
$Announce_i = \{\perp, Agree, Disagree\}$

$$D = \{\langle q, pay_i(c), \delta_{q'}\rangle \mid q_1 = \perp, q_1' = c, j \neq 1 \ \Rightarrow \ q_i = q_i'\}$$
$$\cup\{\langle q, coin_{i-1}(c), \delta_{q'}\rangle \mid q_2 = \perp, q_2' = c, j \neq 2 \ \Rightarrow \ q_i = q_i''\}$$
$$\cup\{\langle q, flip_i, \tfrac{1}{2}\delta_{q'} + \tfrac{1}{2}\delta_{q''}\rangle \mid q_3 = \perp, q_3' = H, q_3'' = T, j \neq 3 \ \Rightarrow \ q_i = q_i' = q_i''\}$$
$$\cup\{\langle q, coin_i(c), \delta_{q'}\rangle \mid q_3 = c \neq \perp, q_4 = F, q_4 = T, j \neq 3, 4 \ \Rightarrow \ q_i = q_i'\}$$
$$\cup\{\langle q, compare_i, \delta_{q'}\rangle \mid q_2 = q_3 \neq \perp, q_5 = \perp, q_5' = T, j \neq 5 \ \Rightarrow \ q_j = q_j'\}$$
$$\cup\{\langle q, compare_i, \delta_{q'}\rangle \mid \perp \neq q_2 \neq q_3 \neq \perp, q_5 = \perp, q_5' = F, j \neq 5 \ \Rightarrow \ q_j = q_j'\}$$
$$\cup\{\langle q, announce_i(c), \delta_{q'}\rangle \mid q_1 = F, q_5 = T, q_6' = \perp, c = q_6' = Agree, j \neq 6 \ \Rightarrow \ q_j = q_j'\}$$
$$\cup\{\langle q, announce_i(c), \delta_{q'}\rangle \mid q_1 = T, q_5 = T, q_6' = \perp, c = q_6' = Disagree, j \neq 6 \ \Rightarrow \ q_j = q_j'\}$$
$$\cup\{\langle q, announce_i(c), \delta_{q'}\rangle \mid q_1 = F, q_5 = F, q_6' = \perp, c = q_6' = Disagree, j \neq 6 \ \Rightarrow \ q_j = q_j'\}$$
$$\cup\{\langle q, announce_i(c), \delta_{q'}\rangle \mid q_1 = T, q_5 = F, q_6' = \perp, c = q_6' = Agree, j \neq 6 \ \Rightarrow \ q_j = q_j'\}$$

Fig 3: The transition relation for Cryptographer i

Our model of the Dining Cryptographers is the composition

[10] Here again, we are using arithmetic modulo 3.

$$DC = Master \parallel C_0 \parallel C_1 \parallel C_2,$$

of the Master with the cryptographers. Note that all actions except the announcements are hidden. Indeed, by definition, the outputs of one component that are inputs of another become hidden in the composition of the two: the Master has no input actions, its outputs are the inputs to the cryptographers, while each cryptographer provides the input for the "next" cryptographer (the one to the right, numbering in the counterclockwise direction). Since all input actions are hidden, DC is a closed task-PIOA.

A prototypical task schedule for this combined process is:

$$choose\_payer, pay_0, pay_1, pay_2, flip_0, flip_1, flip_2, coin_0, coin_1, coin_2,$$
$$compare_0, compare_1, compare_2, announce_0, announce_1, announce_2$$

8 Proving the Protocol Secure

To say DC is secure, we mean that, if one of the cryptographers is paying, then neither of the non-paying cryptographers can tell which of the other cryptographers is paying, based on the probability distributions of their announcements.

Our approach to the proof is first to assume the Master is deterministic, and to show the anonymity of the paying cryptographer from the perspective of the other cryptographers. We then use this result to prove the anonymity of the paying cryptographer in the case of a probabilistic Master who chooses a cryptographer to pay.

8.1 A Deterministic Master

The anonymity guarantee of the Dining Cryptographers – that, if one of the cryptographers is paying for dinner, then neither of the two non-payers can distinguish the behaviors of the remaining cryptographers, can be proved by showing that neither non-paying cryptographer can see a difference in the distributions on the announcements of the other two cryptographers. To accomplish this, we define a deterministic Master using the distribution $r \cdot \delta_{\langle Master, \perp_{Q_0} \rangle} + \sum_i r_i \delta_{\langle Crypto_i, \perp_{Q_0} \rangle}$ satisfying $r_i = 1$ and $r = 0 = r_j$ for $j \neq i$, that specifies a Master $\mathcal{M}_i$ that selects $Crypto_i$, $i = 0, 1$ or 2 as the payer. Let

$$\mathcal{A}_i = \mathcal{M}_i \parallel C_0 \parallel C_1 \parallel C_2 \tag{1}$$

denote the process with Master $\mathcal{M}_i$, $i = 0, 1, 2$. We also let $\mathcal{S}$ denote the set of task schedules.

Proposition 2. *If all cryptographers use fair coins, then the Dining Cryptographers Protocol is deterministically secure for non-paying cryptographers:*

$$(\forall \rho \in \mathcal{S})\ \mathsf{tdist}^{\rho}_{A_i}(\mathcal{A}_{i+1}) = \mathsf{tdist}^{\rho}_{A_i}(\mathcal{A}_{i-1})\ (\forall i = 0, 1, 2).^{11}$$

Thus, for each i and each task schedule ρ, the trace distribution of announcements that $Crypto_i$ sees when the Master selects $Crypto_{i+1}$ are the same as the trace distribution he sees when the Master selects $Crypto_{i-1}$.

[11] We are again counting mod 3, since there are three cryptographers.

Proof. Without loss of generality, we also assume $i = 0$ (so $Crypto_0$ is not paying). The proof proceeds by defining a mapping from $\phi\colon \mathcal{A}_1 \to \mathcal{A}_2$ that simply reverses the roles of $Crypto_1$ and $Crypto_2$ and reverses the values of the received messages, $pay_1(c)$ and $pay_2(c)$.

More formally, we wish to apply Theorem 1. Let $\mathcal{T}$ denote the set of tasks of DC, and let $Q_{\mathcal{A}_i} = Q_M \times \prod_{i=0}^{2} Q_i$ denote the state space of $\mathcal{A}_i$, $i = 1, 2$, where $Q_M = \{Master, Crypto_i \mid i = 0, 1, 2\} \times \prod_{j=0}^{2}\{\bot, T\}$ is the state space of the Master and Q_j is the state space of the j^{th} cryptographer.

Define $\phi\colon Q_{\mathcal{A}_1} \to Q_{\mathcal{A}_2}$ by $\phi(Crypto_1) = Crypto_2$, $\phi(Crypto_2) = Crypto_1$, and ϕ is the identity on all other states. Next, define $\phi\colon \mathsf{Act}_1 \to \mathsf{Act}_2$ by $\phi(pay_1(c)) = pay_2(c)$, and $\phi(pay_2(c)) = pay_1(c)$. On all other actions, ϕ is the identity. We show $\phi\colon \mathcal{A}_1 \to \mathcal{A}_2$ is a Task-PIOA map satisfying the hypothesis of Theorem 1.

We begin by checking conditions (i) and (ii) of Definition 5. For (i), the crucial points are first, what happens to $pay_1(T)$, which is enabled in $\langle Crypto_1, \bot\rangle$. But $\phi(pay_1(T)) = pay_2(T)$, which is enabled in $\langle Crypto_2, \bot\rangle$ in $\mathcal{A}_2$. Likewise, $pay_2(F)$ is enabled in $\langle Crypto_1, \bot\rangle$ in $\mathcal{A}_1$, while $\phi(pay_2(F)) = pay_1(F)$, which is enabled in $\langle Crypto_2, \bot\rangle$ in $\mathcal{A}_2$. Next, we have that $\phi(\delta_{Crypto_1}) = \delta_{\phi(Crypto_1)}$, which is the output distribution of the action *choose_payer*; this follows because $\phi(Crypto_1) = Crypto_2$.

The second point is to check that $\phi(\delta_{\langle Crypto_1, T\rangle}) = \delta_{\langle\phi(Crypto_1),\phi(T)\rangle}$, but this holds because $\phi(Crypto_1) = Crypto_2$ and $\phi(T) = T$. Lastly, the fact that all coins are fair implies that $\mathsf{Disc}(\phi)(\langle \bot, flip_1, \frac{1}{2}\delta_H + \frac{1}{2}\delta_T\rangle) = \langle \bot, flip_2, \frac{1}{2}\delta_H + \frac{1}{2}\delta_T\rangle$, which is the only case in which $\mu_{q,a}$ is not a point mass; for point masses the result follows from the definition of ϕ and from $\mathsf{Disc}(\phi)(\delta_x) = \delta_{\phi(x)}$. This validates the remaining requirement in condition (i) of Definition 5.

Finally, $\mathcal{A}_1$ and $\mathcal{A}_2$ have the same tasks, and ϕ permutes the two actions listed and their corresponding enabling states, so $\phi(T)$ is a task and $\phi(A_T) = A_{\phi(T)}$ for each task T. Thus ϕ is a task-PIOA map satisfying the hypothesis of Theorem 1, so $\mathsf{tdist}(\mathcal{A}_1) \subseteq \mathsf{tdist}(\mathcal{A}_2)$, but since ϕ is its own inverse, the reverse is also true.

Remark 2. We calculate the probability distributions on the announcements: for a non-paying cryptographer, the coins are fair, so the outcome of his own coin is $\frac{1}{2}\delta_H + \frac{1}{2}\delta_T$. The same is true of the coin he sees from his neighbor on the left, so the probability of announcing *Agree* is $\frac{1}{4}\delta_{HH} + \frac{1}{4}\delta_{TT}$, which is $\frac{1}{2}$. Obviously the probability of announcing *Disagree* is also $\frac{1}{2}$. The payer has just been shown to have the same probability distribution on announcements, so the probability is $\frac{1}{2}$ *Agree* and $\frac{1}{2}$ *Disagree* as well.

8.2 The Probabilistic Case

In this section we consider the case of a probabilistic Master – i.e., one that chooses the paying cryptographer probabilistically. This uses the rest of the definition of the Master as given in Figure 1. Thus, the Master process begins with the probabilistic choice $\mathcal{M}_{pr} = r \cdot \delta_{master} + \sum_{i=0}^{2} r_i \delta c_i$ which denotes choosing the Master with probability r and choosing cryptographer i with probability r_i. We call this Master $\mathcal{M}_{pr}$, so our system now is

$$\mathcal{A}_{pr} := \mathcal{M}_{pr} \parallel C_0 \parallel C_1 \parallel C_2.$$

Even though the Master paying is one possibility, the notion of anonymity remains clear: *if* a cryptographer pays, then the identity of the payer should remain unknown to the other cryptographers. Indeed, anonymity is now a *conditional probability*: for a given sample space X and probability measure P, the *conditional probability of A given that B has occurred* is $P(A|B) = \frac{P(A \cap B)}{P(B)}$, provided $P(B) \neq 0$.

The set $\mathcal{E}$ of observable events consists of possible sequences of announcements by the cryptographers. We denote the probability of a sequence $O \in \mathcal{E}$ under the probability measure induced by applying the task sequence ρ to the process $\mathcal{A}_{pr}$ as $\mathsf{Apply}(\delta_{\overline{q}}^{\mathcal{A}_{pr}}, \rho)(O)$, and if O is conditioned on the event O', we denote the probability by $\mathsf{Apply}(\mathcal{A}_{pr}, \rho)(O|O')$.

Theorem 2. *If all cryptographers use unbiased coins, then the Dining Cryptographers is* probabilistically secure*, by which we mean* $(\forall \rho \in \mathcal{S})$ $(\forall O \in \mathcal{E})$

$$\mathit{Apply}(\delta_{\overline{q}}^{\mathcal{A}_{pr}}, \rho)(O|\mathcal{M}_0) = \mathit{Apply}(\delta_{\overline{q}}^{\mathcal{A}_{pr}}, \rho)(O|\mathcal{M}_1) = \mathit{Apply}(\delta_{\overline{q}}^{\mathcal{A}_{pr}}, \rho)(O|\mathcal{M}_2).$$

I.e. the probability distributions on announcements are the same whether the probabilistic Master chooses cryptographer 0, 1 or 2 as the payer.

Proof. There are eight possible announcement sequences under any ρ. If the Master has chosen to pay, any of the four with an even number of *Disagree* announcements occurs, but if one of the cryptographers is paying, then one of the four with an odd number of *Disagree* announcements occurs. We divide $\mathcal{E}$ into the set $\mathcal{E}_E$ of sequences with an even number of *Disagree* announcements, and $\mathcal{E}_O$ of sequences with an odd number of *Disagree* announcements.

For any ρ, Corollary 1 implies $\mathsf{Apply}(\delta_{\overline{q}}^{\mathcal{A}_{pr}}, \rho) = r \cdot \mathsf{Apply}(\delta_{\overline{q}}^{\mathcal{A}_m}, \rho) + \sum_i r_i \cdot \mathsf{Apply}(\delta_{\overline{q}}^{\mathcal{A}_i}, \rho)$, where $\mathcal{A}_i$ is as in Equation 1 and $\mathcal{A}_m$ denotes the case of a deterministic Master choosing himself to pay. We assume $r \neq 0 \neq r_j$ for $j = 0, 1, 2$.

Proposition 2 shows the probability distributions $\mathsf{Apply}(\delta_{\overline{q}}^{\mathcal{A}_i}, \rho)$ all agree on the observable events, and Remark 2 notes the probability that any of the cryptographers announces *Agree* is $\frac{1}{2}$, as is the probability that any of them announces *Disagree*; it's easy to show the same in case the Master is paying. Hence, for any sequence O of announcements and each master $\mathcal{M}_j, j = 0, 1, 2$,

$$\begin{aligned}\mathsf{Apply}(\delta_{\overline{q}}^{\mathcal{A}_{pr}}, \rho)(O \cap \mathcal{M}_j) = {} & \mathsf{Apply}(\delta_{\overline{q}}^{\mathcal{A}_{pr}}, \rho)((O \in \mathcal{E}_E) \cap \mathcal{M}_j) \\ & + \mathsf{Apply}(\delta_{\overline{q}}^{\mathcal{A}_{pr}}, \rho)((O \in \mathcal{E}_O) \cap \mathcal{M}_j).\end{aligned}$$

In case $O \in \mathcal{E}_E$, then the Master is paying, and the only event that can occur is $\mathcal{M}_m$, in which case the probability of any of the possible announcement sequences is $\frac{1}{4}$, since they are equally likely.

The other possibility is that a cryptographer is paying, i.e., $O \in \mathcal{E}_O$. Then

$$\begin{aligned}\mathsf{Apply}(\delta_{\overline{q}}^{\mathcal{A}_{pr}}, \rho)((O \in \mathcal{E}_O) \cap \mathcal{M}_j) &= \left(\sum_i r_i \mathsf{Apply}(\delta_{\overline{q}}^{\mathcal{A}_i}, \rho)\right)((O \in \mathcal{E}_O) \cap \mathcal{M}_j) \\ &= \sum_i r_i \mathsf{Apply}(\delta_{\overline{q}}^{\mathcal{A}_i}, \rho)((O \in \mathcal{E}_O) \cap \mathcal{M}_j) = \frac{r_j}{4}.\end{aligned}$$

Since $\mathsf{Apply}(\delta_{\overline{q}}^{\mathcal{A}_{pr}}, \rho)(\mathcal{M}_j) = r_j \neq 0$, we have $\mathsf{Apply}(\delta_{\overline{q}}^{\mathcal{A}_{pr}}, \rho)(O|\mathcal{M}_j) = 1/4$. This holds for each $j = 0, 1, 2$ for which $r_j \neq 0$. This proves that the paying cryptographer is unknown to the other two.

9 Summary and Future Work

We have presented an alternative derivation of the task-PIOA framework of Canetti, et al [6] based on order theory. The presentation focuses on two of the components needed for task-PIOAs, discrete probability and probabilistic automata theory. Our main result is a proof that task-PIOA maps preserve trace distributions on observable events. We applied our approach by analyzing the Dining Cryptographers Protocol. Our model has much more detail than traditional analyses of the protocol using concurrency, but the point has been to illustrate the use of the task-PIOAs. Of particular note is the modularity of our proof that the protocol meets its security specification – the anonymity of a paying cryptographer from the other cryptographers. We first show this for the case of a deterministic Master, and then use this result to prove the result also holds in the case of a probabilistic Master.

We purposely have avoided the use of cryptographic primitives, and our presentation has been restricted to an example where these aspects are not needed. The addition of encryption would allow an interesting extension of the Dining Cryptographers. Namely, it would support adding a component process one might call *Anonymizer* that provides the service of making the paying cryptographer anonymous even to the Master, one of the original goals of the protocol [9]. This could be accomplished by having the Master's messages to the cryptographers first sent to the *Anonymizer*, which then reroutes the messages using a randomized permutation of the recipients.

We commented in Section 2 that the PIOA approach differs significantly from the approach utilizing labeled Markov processes, as in [10, 16, 19]. The difference concerns the transition relation $D \subseteq Q \times \mathsf{Act} \times \mathsf{Disc}(Q)$ that defines the evolution of processes. For labeled Markov processes, this relation (or the closely related $D \subseteq Q \times \mathsf{Disc}(\mathsf{Act} \times Q)$), assume D generates a function $\Delta\colon Q \times \mathsf{Act} \to \mathsf{Disc}(Q)$ (resp., $\Delta\colon Q \to \mathsf{Disc}(\mathsf{Act} \times Q)$), but in the PIOA approach, Δ is a partial function (cf. comments following Definition 2). The usual method for transforming partial functions into total functions – adjoining a deadlock state and mapping elements of the domain where Δ is not defined to that deadlock state (or, more precisely, point mass at deadlock in the present situation) won't work for PIOAs, because of how the executions of PIOAs are defined (see the footnote to Equation 1).

In future work, we plan to apply our approach to other settings where they are appropriate. Among the areas being scrutinized is the Goldreich-Micali-Wigderson [13] secure multiparty computation protocol, where we also anticipate using the universal composability result for oblivious transfer from [6]. The GMW protocol works by composing a number of simpler protocols, including oblivious transfer and communication mechanisms similar to those among the Dining Cryptographers. The work presented in this paper is thus the first step in that path.

References

1. Abadi, M. and A. Gordon, A calculus for cryptographic protocols: The spi calculus, Information and Computation 148 (1999), pp. 1–70.

2. Abramsky, S. and A. Jung, Domain theory, in: S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, Handbook of Logic in Computer Science, volume 3, pages 1–168. Clarendon Press, 1994.
3. de Alfaro, L., T. Henzinger and R. Jhala, Compositional methods for probabilistic systems, Lecture Notes In Computer Science 2154 (2001), pp. 351–365.
4. Bhargava, M. and C. Palamidessi, Probabilistic anonymity, Proceedings of CONCUR 2005, LNCS 3653 (2005), pp. 171–185.
5. Canetti, R., Universally Composable Security: A New Paradigm for Cryptographic Protocols, Available online at URL: `http://eprint.iacr.org/2000/067`
6. Canetti, R., N. Lynch, et al., Using Probabilistic I/O Automata to Analyze an Oblivious Transfer Protocol, preprint `http://people.csail.mit.edu/lcheung/task-pioa/task-PIOA-TR.pdf`.
7. Chatzikokalakis, K. and C. Palamidessi, Making Random Choices Invisible to the Scheduler, Lecture Notes in Computer Science **4703**, Springer-Verlag, 2007, pp. 42–58.
8. Chatzikokalakis, K., C. Palamidessi and P. Panangaden, Anonymity protocols as noisy channels, Information and Computation 206 (2008), pp. 378–401.
9. Chaum, D. , The dining cryptographers problem: Unconditional sender and recipient untraceability, Journal of Cryptology 1 (1988), pp. 65-75.
10. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Approximating Labeled Markov Processes. *Information and Computation*, 184(1):160–200, 2003.
11. Garcia, F., P. van Rossum and A. Sokolova, Probabilistic anonymity and admissible schedulers, Jun2, 2007, arXiv:0706.1019, `http://eprintweb.org/S/authors/All/so/Sokolova`
12. Goldreich, O., "Secure Multi-party Computation," Available online at URL : `http://www.wisdom.weizmann.ac.il/~oded/pp.html`
13. Goldreich, O., S. Micali and A. Wigderson, Proofs that yield nothing but their validity, or All languages in NP have Zero-knowledge proofs. JACM **38** (1991), pp 691–729.
14. Hasuo, I. and Y. Kawabe, Probabilistic anonymity via coalgebraic simulations, in: *Proceedings of Programming Languages and Systems*, LNCS **4421**(2007), pp. 379-394
15. Jones, C., Probabilistic non-determinism, University of Edinburgh, Edinburgh, Scotland, 1992
16. K.G. Larsen and A. Skou. Bisimulation through Probabilistic Testing. *Information and Computation*, 94(1):1–28, 1991.
17. Lincoln, P. J. C. Mitchell, M. Mitchell and A. Scedrov, A Probabilistic Poly-Time Framework for Protocol Analysis, ACM Conference on Computer and Communications Security (1998), pp. 112-121.
18. Mislove, M.. Nondeterminism and probabilistic choice: obeying the laws, Proceedings of CONCUR 2000, Lecture Notes in Computer Science **1877** (2000), pp. 350 364.
19. Mislove, M., D. Pavlovic and J. Worrell, Labelled Markov Processes as Generalized Stochastic Relations, Electronic Notes Theoretical Computer Science **172** (2007), pp: 459–478.
20. Mitchell, J., A. Ramanathan, A. Scedrov, V. Teague, A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols, Theoretical Computer Science 353 (2006), pp. 118–164.
21. Schneider, S. and A. Sidiropoulos, CSP and anonymity, in Proceedings of European Symposium on Research in Computer Security (ESORICS), LNCS 1146 (1996), pp. 198-218, `http://dx.doi.org/10.1007/3-540-61770-1_38`

22. Segala, R., Modeling and Verification of Randomized Distributed Real-time Systems, PhD Thesis (1995), MIT Technical Report MIT/LCS/TR-676.
23. Segala, R. and N. Lynch, Probabilistic simulations for probabilistic processes, Nordic Journal of Computing **2** (1995), 250–273.

A Appendix - Proofs

We present the proofs of selected results not proved in the body of the paper.

Proposition 1 Let $\mu = \sum_\alpha \mu(\alpha)\delta_\alpha \in \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{A}))$, then

$$\mu \leq \mathsf{Apply}(\sum_\alpha \mu(\alpha)\delta_\alpha, \rho) = \sum_\alpha \mu(\alpha)\mathsf{Apply}(\delta_\alpha, \rho).$$

In particular, $\mathsf{Apply}(\mu, \rho)$ is well-defined for any task sequence $\rho = T_1 \cdots T_n$.

Proof. If $\rho = \lambda$ the empty sequence, then $\mathsf{Apply}(\mu, \rho) = \mu$, so the result is trivial, while if $\rho = T$ is a single task, then

$$\begin{aligned}\mathsf{Apply}(\mu, T) &= \sum_{\alpha \notin A_T} \mu(\alpha)\delta_\alpha + \sum_{\alpha \in A_T} \mu(\alpha)\left(\sum_s \mu_{\mathsf{lstate}(\alpha),a}(s)\delta_{\alpha a s}\right) \qquad (2)\\ &= \sum_{\alpha \notin A_T} \mu(\alpha)\mathsf{Apply}(\delta_\alpha, T) + \sum_{\alpha \in A_T} \mu(\alpha)\mathsf{Apply}(\delta_\alpha, T)\\ &= \sum_\alpha \mu(\alpha)\mathsf{Apply}(\delta_\alpha, T).\end{aligned}$$

The fact that $\mu \leq \mathsf{Apply}(\mu, T)$ is obvious from Equation 2.

Next, if $\rho = \rho' T$ is finite, then

$$\begin{aligned}\mathsf{Apply}(\mu, \rho) &= \mathsf{Apply}(\mathsf{Apply}(\mu, \rho'), T) = \mathsf{Apply}(\sum_\alpha \mu(\alpha)\mathsf{Apply}(\delta_\alpha, \rho'), T)\\ &= \sum_\alpha \mu(\alpha)\mathsf{Apply}(\mathsf{Apply}(\delta_\alpha, \rho'), T)\end{aligned}$$

Assuming by induction that $\mu \leq \mathsf{Apply}(\mu, \rho')$, then $\mu \leq \mathsf{Apply}(\mu, \rho)$ follows from the basis step.

Theorem 1 *Let $\phi\colon \mathcal{A}_1 \to \mathcal{A}_2$ be a Task PIOA map of compatible task PIOAs satisfying $\phi(A_T) = A_{\phi(T)}$ for each task $T \in \mathcal{T}_1$. If $A \subseteq \mathsf{Act}_1$ is the set of observable actions, then $\mathsf{Disc}(\phi|^*_A))(\mathsf{tdist}(\mathcal{A})) \subseteq \mathsf{tdist}(\mathcal{A}_2)$.*

Proof. We first show that if $A_i \subseteq \mathsf{Act}_i, i = 1, 2$ are the sets of *observable events* and if $\phi(A_1) \subseteq A_2$, then the following diagram commutes:

$$\begin{array}{ccccc}
\mathsf{Disc}(\mathsf{Exec}^*\mathcal{A}_1) \times \mathcal{T} & \xrightarrow{\mathsf{Apply}} & \mathsf{Disc}(\mathsf{Exec}^*\mathcal{A}_1) & \xrightarrow{\mathsf{Disc}(\mathsf{trace})} & \mathsf{Disc}(A_1^*) \\
\downarrow{\scriptstyle \mathsf{Disc}(\mathsf{Exec}^*\phi)\times\phi} & & \downarrow{\scriptstyle \mathsf{Disc}(\mathsf{Exec}^*\phi)} & & \downarrow{\scriptstyle \mathsf{Disc}(\phi|^*_{A_1})} \\
\mathsf{Disc}(\mathsf{Exec}^*\mathcal{A}_2) \times \mathcal{T} & \xrightarrow[\mathsf{Apply}]{} & \mathsf{Disc}(\mathsf{Exec}^*\mathcal{A}_2) & \xrightarrow[\mathsf{Disc}(\mathsf{trace})]{} & \mathsf{Disc}(A_2^*),
\end{array}$$

where $\phi^*\colon A_1^* \to A_2^*$ is given by $\phi^*(a_0 \cdots a_n) = \phi(a_0) \cdots \phi(a_n)$. Indeed, the commutativity of the left square is a diagram chase, with one twist: we need $\phi(T) \in \mathcal{T}_2$ for each $T \in \mathcal{T}_1$: Indeed, if $\sum_i r_i \delta_{\alpha_i} \in \mathsf{Disc}(\mathsf{Exec}^* \mathcal{A}_1)$ and $T \in \mathcal{T}_1$, then

$$\begin{aligned}
&\mathsf{Disc}(\mathsf{Exec}^*\phi)(\mathsf{Apply}(\sum_i r_i \delta_{\alpha_i}, T)) = \\
&= \mathsf{Disc}(\mathsf{Exec}^*\phi)\left(\sum_{\alpha_i \notin A_T} r_i \delta_{\alpha_i} + \sum_{\alpha_i \in A_T} r_i \sum_{q \in Q_1} \mu_{\mathsf{lstate}(\alpha_i), a_T}(q) \delta_{\alpha_i a_T q}\right) \\
&\overset{1}{=} \sum_{\alpha_i \notin A_T} r_i \delta_{\mathsf{Exec}^*\phi(\alpha_i)} + \sum_{\alpha_i \in A_T} r_i \sum_{q \in Q_1} \mu_{\mathsf{lstate}(\mathsf{Exec}^*\phi(\alpha_i)), \phi(a_T)}(\phi(q)) \delta_{\mathsf{Exec}^*\phi(\alpha_i a_T q)} \\
&\overset{2}{=} \sum_{\mathsf{Exec}^*\phi(\alpha_i) \notin A_{\phi(T)}} r_i \delta_{\mathsf{Exec}^*\phi(\alpha_i)} + \\
&\qquad \sum_{\mathsf{Exec}^*\phi(\alpha_i) \in A_{\phi(T)}} r_i \sum_{\phi(q) \in Q_2} \mu_{\mathsf{lstate}(\mathsf{Exec}^*\phi(\alpha_i)), \phi(a_T)}(\phi(q)) \delta_{\mathsf{Exec}^*\phi(\alpha_i a_T q)} \\
&= \mathsf{Apply}\left(\mathsf{Disc}(\mathsf{Exec}^*\phi)(\sum_i r_i \delta_{\alpha_i}), \phi(T)\right),
\end{aligned}$$

where $\overset{1}{=}$ follows from the definition of $\mathsf{Disc}(\mathsf{Exec}^*\phi)$, while $\overset{2}{=}$ follows from the assumptions that $\phi(T) \in \mathcal{T}_2$ is a task in $\mathcal{A}_2$, from the assumption that $\phi(A_T) = A_{\phi(T)}$ (to preserve the first summand in each case) and from the calculation that $\mathsf{Exec}^*\phi(\mathsf{lstate}(\alpha_i)) = \mathsf{lstate}(\mathsf{Exec}^*\phi(\alpha_i))$. The right square is then a simple diagram chase.

Now the result follows: Let $\rho = T_1 \cdots T_n \in \mathcal{T}_1^*$ be a task schedule. Then for $\sum_i r_i \delta_{\alpha_i} \in \mathsf{Disc}(\mathsf{Exec}^* \mathcal{A}_1)$, we have

$$\begin{aligned}
&\mathsf{Disc}(\mathsf{Exec}^*\phi)(\mathsf{Apply}(\sum_i r_i \delta_{\alpha_i}, \rho)) = \mathsf{Disc}(\phi^*)(\mathsf{Apply}(\mathsf{Apply}(\sum_i r_i \delta_{\alpha_i}, T_1 \cdots T_{n-1}), T_n))) \\
&\quad = \mathsf{Apply}\left(\mathsf{Disc}(\mathsf{Exec}^*\phi)(\mathsf{Apply}(\sum_i r_i \delta_{\alpha_i}, T_1 \cdots T_{n-1})), \phi(T_n))\right)
\end{aligned}$$

by 2). It then follows by induction on n that

$$\mathsf{Disc}(\mathsf{Exec}^*\phi)(\mathsf{Apply}(\sum_i r_i \delta_{\alpha_i}, \rho)) = \mathsf{Apply}\left(\mathsf{Disc}(\mathsf{Exec}^*\phi)(\sum_i r_i \delta_{\alpha_i}), \phi(T_1) \cdots \phi(T_n)\right).$$

Thus,

$$\begin{aligned}
&\mathsf{Disc}(\mathsf{trace}) \circ \mathsf{Disc}(\mathsf{Exec}^*\phi)(\mathsf{Apply}(\sum_i r_i \delta_{\alpha_i}, \rho)) = \\
&\qquad \mathsf{Disc}(\mathsf{trace})\left(\mathsf{Apply}\left(\mathsf{Disc}(\mathsf{Exec}^*\phi)(\sum_i r_i \delta_{\alpha_i}), \phi(T_1) \cdots \phi(T_n)\right)\right).
\end{aligned}$$

But the left square in 2) implies

$$\mathsf{Disc}(\mathsf{trace}) \circ \mathsf{Disc}(\mathsf{Exec}^*\phi)(\mathsf{Apply}(\sum_i r_i \delta_{\alpha_i}, \rho)) = \mathsf{Disc}(\phi|_{A_1}^*)\left(\mathsf{Disc}(\mathsf{trace})(\mathsf{Apply}(\sum_i r_i \delta_{\alpha_i}, \rho))\right),$$

which implies the claim.