Safe Equivalences for Security Properties

Mário S. Alvim[1], Miguel E. Andrés[2], Catuscia Palamidessi[1], and Peter van Rossum[2]

[1] INRIA and LIX, École Polytechnique Palaiseau, France
[2] Institute for Computing and Information Sciences, The Netherlands

Abstract. In the field of Security, process equivalences have been used to characterize various information-hiding properties (for instance secrecy, anonymity and non-interference) based on the principle that a protocol P with a variable x satisfies such property if and only if, for every pair of secrets s_1 and s_2, $P[^{s_1}/_x]$ is equivalent to $P[^{s_2}/_x]$. We argue that, in the presence of nondeterminism, the above principle relies on the assumption that the scheduler "works for the benefit of the protocol", and this is usually not a safe assumption. Non-safe equivalences, in this sense, include complete-trace equivalence and bisimulation. We present a formalism in which we can specify admissible schedulers and, correspondingly, safe versions of these equivalences. We prove that safe bisimulation is still a congruence. Finally, we show that safe equivalences can be used to establish information-hiding properties.

1 Introduction

One of the fundamental problems in computer security is the protection from information leaks, namely how to make sure that a system does not reveal, by observations that can be made during the execution, some information that we wish to maintain secret.

One way to prevent an attacker to infer the secret from the observables is to create *noise*, namely to make sure that for every execution in which a given secret produces a certain observable, there is at least another execution in which a different secret produces the same observable. In practice this is often done by using randomization, see for instance the DCNet [10] and the Crowds [23] protocols.

In the literature about the foundations of Computer Security, however, the quantitative aspects are often abstracted away, and probabilistic behavior is replaced by nondeterministic behavior. Correspondingly, there have been various approaches in which information-hiding properties are expressed in terms of equivalences based on nondeterminism, especially in a concurrent setting. For instance, [24] defines *anonymity* as follows[1]: A protocol S is anonymous if, for every pair of culprits a and b, $S[^a/_x]$ and $S[^b/_x]$ produce the same observable traces. A similar definition is given in [1] for *secrecy*, with the difference that $S[^a/_x]$ and $S[^b/_x]$ are required to be bisimilar. In [13], an electoral system S preserves the *confidentiality of the vote* if for any voters v and w, the observable behavior of S is the same if we swap the votes of v and w. Namely, $S[^a/_v\ |^b\ /_w] \sim S[^b/_v\ |^a\ /_w]$, where $\sim$ represents bisimilarity.

[1] The actual definition of [24] is more complicated, but the spirit is the same.

C.S. Calude and V. Sassone (Eds.): TCS 2010, IFIP AICT 323, pp. 55–70, 2010.

These proposals are based on the implicit assumption that *all the nondeterministic executions present in the specification of S will always be possible under every implementation of S*. Or at least, that the adversary will believe so. In concurrency, however, as argued in [8], nondeterminism has a rather different meaning: if a specification S contains some nondeterministic alternatives, typically it is because we want to abstract from specific implementations, such as the scheduling policy. A specification is considered correct, with respect to some property, if every alternative satisfies the property. Correspondingly, an implementation is considered correct if all executions are among those possible in the specification, i.e. if the implementation is a refinement of the specification. There is no expectation that the implementation will actually make possible all the alternatives indicated by the specification.

We argue that the use of nondeterminism in concurrency corresponds to a *demonic* view: the scheduler, i.e. the entity that will decide which alternative to select, may try to choose the worst alternative. Hence we need to make sure that "all alternatives are good", i.e. satisfy the intended property. In the above mentioned approaches to the formalization of security properties, on the contrary, the interpretation of nondeterminism is *angelic*: the scheduler is expected to actually help the protocol to confuse the adversary and thus protect the secret information.

There is another issue, orthogonal to the angelic/demonic dichotomy, but relevant for the achievement of security properties: the scheduler *should not be able to make its choices dependent on the secret*, or else nearly every protocol would be insecure, i.e. the scheduler would always be able to leak the secret to an external observer (for instance by producing different interleavings of the observables, depending on the secret). This remark has been made several times already, and several approaches have been proposed to cope with the problem of full-information scheduler (aka almighty, omniscient, clairvoyant, etc.), see for example [6,7,9,8,3].

The risk of a naive use of nondeterminism to specify a security property, is not only that it may rely on an implicit assumption that the scheduler behaves angelically, but also that it is clairvoyant (fully-informed), i.e. that it peeks at the secrets (that it is not supposed to be able to see) to achieve its angelic strategy.

Example 1. Consider the following system, in a CCS-like syntax: $S \stackrel{\text{def}}{=} (c)(A \parallel H_1 \parallel H_2 \parallel Corr)$, with $A \stackrel{\text{def}}{=} \overline{c}\langle sec\rangle$, $H_1 \stackrel{\text{def}}{=} c(s).\overline{out}\langle a\rangle$, $H_2 \stackrel{\text{def}}{=} c(s).\overline{out}\langle b\rangle$, $Corr \stackrel{\text{def}}{=} c(s).\overline{out}\langle s\rangle$. Here $\parallel$ is the parallel operator, $\overline{c}\langle sec\rangle$ is a process that sends sec on channel c, $c(s).P$ is a process that receives s on channel c and then continues as P, and (c) is the restriction operator, enforcing synchronization on c. The name sec represents a secret.

It is easy to see that we have $S\left[{}^{a}/_{sec}\right] \sim S\left[{}^{b}/_{sec}\right]$. Note that, in order to simulate the third branch in $S\left[{}^{a}/_{sec}\right]$, the process $S\left[{}^{b}/_{sec}\right]$ needs to select its first branch. Viceversa, in order to simulate the third branch in $S\left[{}^{b}/_{sec}\right]$, the process $S\left[{}^{a}/_{sec}\right]$ needs to select its second branch. This means that, in order to achieve bisimulation, the scheduler needs to know the secret, and change its choice accordingly.

This example shows a system that intuitively is not secure, because the third component, $Corr$, reveals whatever secret it receives. However, according to the equivalence-based notions of security discussed above, *it is secure*. But it is secure thanks to a

scheduler that angelically helps the system to protect the secret, and it does so by making its choices dependent on the secret. We consider these assumptions on the scheduler excessively strong.

We do not claim, however, that we should rule out the use of angelic nondeterminism in security: on the contrary, angelic nondeterminism can be a powerful specification concept. We only advocate a cautious use of this notion. In particular, it should not be used in a context in which the scheduler may be in collusion with the attacker. The goal of this paper is to define a framework in which we can combine both angelic and demonic nondeterminism in a setting in which also probabilistic behavior may be present, and in a context in which the scheduler is restricted (i.e. not fully-informed). We define "safe" variant of typical equivalence relations (complete traces and bisimulation), and we show how to use them to characterize information-hiding properties.

1.1 Contribution

The main novelties of our work can be articulated as follows:

- We propose a formalism for concurrent systems which accounts for both probabilistic and nondeterministic behaviour, and in which the latter is of two kinds: *global* and *local*. The first represents the possible interleavings produced by the parallel components, which may be influenced by the attacker. The second is associated to the possible choices internal to each component, which may depend on the secrets or other unknown parameters, not controlled by the attacker. Correspondingly, we split the scheduler in two constituents: global and local. The latter is actually a tuple of local schedulers, one for each component of the system.
- We propose a notion of *admissible scheduler* for the above systems, in which the global constituent is not allowed to see the secrets, and each local constituent is not allowed to see any information about the other components. We then generalize the standard definition of strong (probabilistic) information hiding (such as no-interference and strong anonymity) to the case in which also nondeterminism is present, under the assumption that the schedulers are admissible.
- We use admissible schedulers to define safe versions of complete-trace equivalence and bisimilarity especially tuned for security (in this paper we often refer to complete traces as simply traces). This means that we account for the possibility that the global constituent of the scheduler is in collusion with the attacker, and therefore does not necessarily help the system to obfuscate the secret. We show that the latter is still a congruence, like in the classical case.
- We finally show that our notions of safe trace equivalence and bisimilarity imply strong information hiding in the above sense.

2 Probabilistic Automata

In this section we gather preliminary notions and results related to probabilistic automata [26,25].

A function $\mu\colon Q \to [0,1]$ is a *discrete probability distribution* on a set Q if the support of μ is countable and $\sum_{q\in Q}\mu(q) = 1$. The set of all discrete probability distributions on Q is denoted by $\mathcal{D}(Q)$.

A *probabilistic automaton* is a quadruple $M = (Q, \Sigma, \hat{q}, \alpha)$ where Q is a countable set of *states*, Σ a finite set of *actions*, $\hat{q}$ the *initial* state, and α is a *transition function* $\alpha : Q \to \mathcal{P}(\Sigma\times\mathcal{D}(Q))$. Here $\mathcal{P}(X)$ is the set of all finite subsets of X. If $\alpha(q) = \emptyset$ then q is a *terminal* state. We write $q \xrightarrow{a} \mu$ for $(a,\mu) \in \alpha(q)$. Moreover, we write $q \xrightarrow{a} r$ whenever $q \xrightarrow{a} \mu$ and $\mu(r) > 0$. A *fully probabilistic automaton* is a probabilistic automaton satisfying $|\alpha(q)| \leq 1$ for all states. In case $\alpha(q) \neq \emptyset$ in a fully probabilistic automaton, we will overload notation and use $\alpha(q)$ to denote the distribution outgoing from q. A *path* in a probabilistic automaton is a sequence $\sigma = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots$ where $q_i \in Q$, $a_i \in \Sigma$ and $q_i \xrightarrow{a_{i+1}} q_{i+1}$. A path can be *finite* in which case it ends with a state. A path is *complete* if it is either infinite or finite ending in a terminal state. Given a path σ, $first(\sigma)$ denotes its first state, and if σ is finite then $last(\sigma)$ denotes its last state. Let $\mathrm{Paths}_q(M)$ denote the set of all paths, $\mathrm{Paths}^\star_q(M)$ the set of all finite paths, and $\mathrm{CPaths}_q(M)$ the set of all complete paths of an automaton M, starting from the state q. We will omit q if $q = \hat{q}$. Paths are ordered by the prefix relation, which we denote by $\leq$. The *trace* of a path is the sequence of actions in $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ obtained by removing the states, hence for the above path σ we have $trace(\sigma) = a_1 a_2 \ldots$. We denote by $Traces(M)$ the complete traces of M, i.e. $Traces(M) \stackrel{\text{def}}{=} \{trace(\sigma) \mid \sigma \in \mathrm{CPaths}(M)\}$. If $\Sigma' \subseteq \Sigma$, then $trace_{\Sigma'}(\sigma)$ is the projection of $trace(\sigma)$ on the elements of Σ'.

Let $M = (Q, \Sigma, \hat{q}, \alpha)$ be a (fully) probabilistic automaton, $q \in Q$ a state, and let $\sigma \in \mathrm{Paths}^\star_q(M)$ be a finite path starting in q. The *cone* generated by σ is the set of complete paths $\langle\sigma\rangle = \{\sigma' \in \mathrm{CPaths}_q(M) \mid \sigma \leq \sigma'\}$. Given a fully probabilistic automaton $M = (Q, \Sigma, \hat{q}, \alpha)$ and a state q, we can calculate the *probability value*, denoted by $\mathbf{P}_q(\sigma)$, of any finite path σ starting in q as follows: $\mathbf{P}_q(q) = 1$ and $\mathbf{P}_q(\sigma \xrightarrow{a} q') = \mathbf{P}_q(\sigma)\cdot\mu(q')$, where $last(\sigma) \xrightarrow{a} \mu$. Let $\Omega_q \stackrel{\text{def}}{=} \mathrm{CPaths}_q(M)$ be the sample space, and let $\mathcal{F}_q$ be the smallest σ-algebra generated by the cones. Then $\mathbf{P}_q$ induces a unique *probability measure* on $\mathcal{F}_q$ (which we will also denote by $\mathbf{P}_q$) such that $\mathbf{P}_q(\langle\sigma\rangle) = \mathbf{P}_q(\sigma)$ for every finite path σ starting in q. For $q = \hat{q}$ we write $\mathbf{P}$ instead of $\mathbf{P}_{\hat{q}}$.

A scheduler for a probabilistic automaton M is a function $\zeta\colon \mathrm{Paths}^\star(M) \to (\Sigma \times \mathcal{D}(Q) \cup \{\bot\})$ such that for all finite path σ, if $\alpha(last(\sigma)) \neq \emptyset$ then $\zeta(\sigma) \in \alpha(last(\sigma))$, and $\zeta(\sigma) = \bot$ otherwise. Hence, a scheduler ζ selects one of the available transitions in each state, and determines therefore a fully probabilistic automaton, obtained by pruning from M the alternatives that are not chosen by ζ. A scheduler is history dependent since it takes into account the path and not only the current state. It may be partial, i.e. it may halt the execution at any time [2].

3 Systems

In this section we describe the kind of systems we are dealing with. We start by introducing a variant of probabilistic automata, that we call *Tagged Probabilistic Automata*

[2] In this paper, however, we will consider only total schedulers, to be more in line with the standard semantics of CCS.

(TPA). These systems are parallel compositions of probabilistic processes, called *components*. Each component is equipped with a unique identifier, called *tag*. Whenever a component (or a pair of components in case of synchronization) makes a step, the corresponding transition will be decorated with the associated tag (or pair of tags).

Similar systems have been already introduced in [3]. The main differences are that here the components may contain nondetermism, and a secret can label any transition.

3.1 Tagged Probabilistic Automata

We now formalize the notion of TPA.

Definition 1. *A* Tagged Probabilistic Automaton *is a tuple* $(Q, L, \Sigma, \hat{q}, \alpha)$*, where* Q *is a set of* states*,* L *is a set of* tags*,* Σ *is a set of* actions*,* $\hat{q} \in Q$ *is the* initial state*,* $\alpha \colon Q \to \mathcal{P}(L \times \Sigma \times \mathcal{D}(Q))$ *is a* transition function*.*

In the following we write $q \xrightarrow{l:a} \mu$ for $(\ell, a, \mu) \in \alpha(q)$, and we use $enab(q)$ to denote the tags of the components that are enabled to make a transition. Namely, $enab(q) \stackrel{\text{def}}{=} \{\ell \in L \mid \textit{there exists } a \in \Sigma, \mu \in \mathcal{D}(Q) \textit{ such that } q \xrightarrow{l:a} \mu\}$. In these systems, we can decompose the scheduler in two: a *global scheduler*, which decides which component or pair of components makes the move next, and a *local scheduler*, which solves the internal nondeterminism of the selected component.

We assume that the local scheduler can only select enabled transitions, and that the global scheduler can only select enabled components. This means that the execution does not stop unless all components are blocked. This is in line with the tradition of process algebra and of Markov Decision Processes, but contrasts with that of Probabilistic Automata [26]. However, the results in this paper do not depend on this assumption.

Definition 2. *Let* $M = (Q, L, \Sigma, \hat{q}, \alpha)$ *be a Tagged Probabilistic Automaton.*

- *A global scheduler for* M *is a function* $\zeta \colon \mathrm{Paths}^\star(M) \to (L \cup \{\bot\})$ *such that for all finite paths* σ*, if* $enab(last(\sigma)) \neq \emptyset$ *then* $\zeta(\sigma) \in enab(last(\sigma))$*, and* $\zeta(\sigma) = \bot$ *otherwise.*
- *A local scheduler for* M *is a function* $\xi \colon \mathrm{Paths}^\star(M) \to (L \times \Sigma \times \mathcal{D}(Q) \cup \{\bot\})$ *such that, for all finite paths* σ*, if* $\alpha(last(\sigma)) \neq \emptyset$ *then* $\xi(\sigma) \in \alpha(last(\sigma))$*, and* $\xi(\sigma) = \bot$ *otherwise.*
- *A global scheduler* ζ *and a local scheduler* ξ *for* M *are* compatible *if, for all finite paths* σ*,* $\xi(\sigma) = (\ell, a, \mu)$ *implies* $\zeta(\sigma) = \ell$*, and* $\xi(\sigma) = \bot$ *implies* $\zeta(\sigma) = \bot$*.*
- *A scheduler is a pair* (ζ, ξ) *of compatible global and local schedulers.*

3.2 Components

We are going to use a simple probabilistic process calculus (a sort of probabilistic version of CCS [20,21]) to specify the components.

We assume a set of *actions* or *channel names* Σ with elements $a, a_1, a_2, \cdots$, including the special symbol τ denoting a *silent step*. Except τ, each action a has a co-action $\bar{a} \in \Sigma$ and we assume $\bar{\bar{a}} = a$. Components are specified by the following grammar:

$$q \quad ::= \quad 0 \quad | \quad a.q \quad | \quad q_1 + q_2 \quad | \quad \sum_i p_i : q_i \quad | \quad q_1 | q_2 \quad | \quad (a)q \quad | \quad A$$

The constructs 0, $a.q$, $q_1 + q_2$, $q_1|q_2$ and $(a)q$ represent termination, prefixing, nondeterministic choice, parallel composition, and the restriction operator, respectively. $\sum_i p_i : q_i$ is a probabilistic choice, where p_i represents the probability of the i-th branch and must satisfy $0 \leq p_i \leq 1$ and $\sum_i p_i = 1$. The process call A is a simple process identifier. For each identifier, we assume a corresponding unique process declaration of the form $A \stackrel{\text{def}}{=} q$. The idea is that, whenever A is executed, it triggers the execution of q. Note that q can contain A or another process identifier, which means that our language allows (mutual) recursion. We will denote by $fn(q)$ the *free channel names* occurring in q, i.e. the channel names not bound by a restriction operator.

Components' semantics: The operational semantics consists of probabilistic transitions of the form $q \xrightarrow{a} \mu$ where $q \in Q$ is a process, $a \in \Sigma$ is an action and $\mu \in \mathcal{D}(Q)$ is a distribution on processes. They are specified by the following rules:

$$\text{PRF}\ \frac{}{a.q \xrightarrow{a} \delta_q} \qquad \text{NDT}\ \frac{q_1 \xrightarrow{a} \mu}{q_1 + q_2 \xrightarrow{a} \mu}$$

$$\text{PRB}\ \frac{}{\sum_i p_i : q_i \xrightarrow{\tau} \sum_i p_i \cdot \delta_{q_i}} \qquad \text{PAR}\ \frac{q_1 \xrightarrow{a} \mu}{q_1 \mid q_2 \xrightarrow{a} \mu \mid q_2}$$

$$\text{CALL}\ \frac{q \xrightarrow{a} \mu}{A \xrightarrow{a} \mu}\ \text{if } A \stackrel{\text{def}}{=} q \qquad \text{COM}\ \frac{q_1 \xrightarrow{a} \delta_{r_1} \quad q_2 \xrightarrow{\bar{a}} \delta_{r_2}}{q_1 \mid q_2 \xrightarrow{\tau} \delta_{r_1|r_2}} \qquad \text{RST}\ \frac{q \xrightarrow{a} \mu}{(b)q \xrightarrow{a} (b)\mu}\ a,\bar{a} \neq b$$

We assume also the symmetric versions of the rules NDT, PAR and COM. The symbol δ_q is the delta of Dirac, which assigns probability 1 to q and 0 to all other processes. The symbol $\sum_i$ is the summation on distributions. Namely, $\sum_i p_i \cdot \mu_i$ is the distribution μ such that $\mu(x) = \sum_i p_i \cdot \mu_i(x)$. The notation $\mu \mid q$ represents the distribution μ' such that $\mu'(r) = \mu(q')$ if $r = q' \mid q$, and $\mu'(r) = 0$ otherwise. Similarly, $(b)\mu$ represents the distribution μ' such that $\mu'(q) = \mu(q')$ if $q = (b)q'$, and $\mu'(q) = 0$ otherwise.

3.3 Systems

A system has the form $(A)\ q_1 \parallel q_2 \parallel \cdots \parallel q_n$, where the q_i's are components and $A \subseteq \Sigma$. The restriction on A enforces synchronization on the channel names belonging to A, in accordance with the CCS spirit.

Systems' semantics The semantics of a system gives rise to a TPA, where the states are terms representing systems during their evolution. A transition now is of the form $q \xrightarrow{\ell:a} \mu$ where $a \in \Sigma$, $\mu \in \mathcal{D}(Q)$, and $\ell \in L$ is either the tag of the component which makes the move, or a (unordered) pair of tags representing the two partners of a synchronization. We can simply define L as $L = I \cup I^2$ where $I = \{1, 2, \ldots, n\}$.

$$\text{Interleaving}\ \frac{q_i \xrightarrow{a} \sum_j p_j \cdot \delta_{q_{ij}}}{(A)\ q_1 \parallel \cdots \parallel q_i \parallel \cdots \parallel q_j \parallel \cdots \parallel q_n \xrightarrow{i:a} \sum_j p_j \cdot \delta_{(A)q_1 \parallel \cdots \parallel q_{ij} \parallel \cdots \parallel q_n}}\ a \notin A$$

where i is the tag indicating that the component i is making the step. Note that we assume that probabilistic choices are finite. This implies that every transition $q \xrightarrow{\ell:a} \mu$ can be written $q \xrightarrow{\ell:a} \sum_i p_i \cdot \delta_{q_i}$, and justifies the notation used in the interleaving rule.

$$\text{Synchronization}\ \frac{q_i \xrightarrow{a} \delta_{q_i'} \qquad q_j \xrightarrow{\bar{a}} \delta_{q_j'}}{(A)\ q_1 \parallel \cdots \parallel q_i \parallel \cdots \parallel q_j \parallel \cdots \parallel q_n \xrightarrow{\{i,j\}:\tau} \delta_{(A)q_1\parallel\cdots\parallel q_i'\parallel\cdots\parallel q_j'\parallel\cdots\parallel q_n}}$$

here $\{i, j\}$ is the tag indicating that the components making the step are i and j. Note that it is an unordered pair. Sometimes we will write i, j instead of $\{i, j\}$, for simplicity.

Example 2. Consider the systems of Example 1. Figures 1(a) and 1(b) show the TPAs of $S\left[{}^{a}/_{sec}\right]$ and of $S\left[{}^{b}/_{sec}\right]$ respectively. For simplicity we do not write the restriction on channels c and out, and the termination symbol 0. We use '$-$' to denote a component that is stuck. The corresponding tags are indicated in the figure with numbers above the components. The set of enabled transitions should be clear from the figures. For instance, we have $enab(S\left[{}^{b}/_{sec}\right]) = \{\{1,2\},\{1,3\},\{1,4\}\}$ and $enab(-\parallel \overline{out}\langle a\rangle \parallel - \parallel - \parallel -) = \{2\}$. The scheduler ζ defined as

$$\zeta(\sigma) \stackrel{\text{def}}{=} \begin{cases} \{1,4\} & if\ \sigma = S\left[{}^{a}/_{sec}\right], \\ 2 & if\ \sigma = S\left[{}^{a}/_{sec}\right] \xrightarrow{1,2:\tau} (-\parallel \overline{out}\langle a\rangle \parallel - \parallel - \parallel -), \\ 3 & if\ \sigma = S\left[{}^{a}/_{sec}\right] \xrightarrow{1,3:\tau} (-\parallel - \parallel \overline{out}\langle b\rangle \parallel - \parallel -), \\ 4 & if\ \sigma = S\left[{}^{a}/_{sec}\right] \xrightarrow{1,4:\tau} (-\parallel - \parallel - \parallel \overline{out}\langle a\rangle \parallel -), \\ \bot & otherwise, \end{cases}$$

is a global scheduler for $S\left[{}^{a}/_{sec}\right]$.

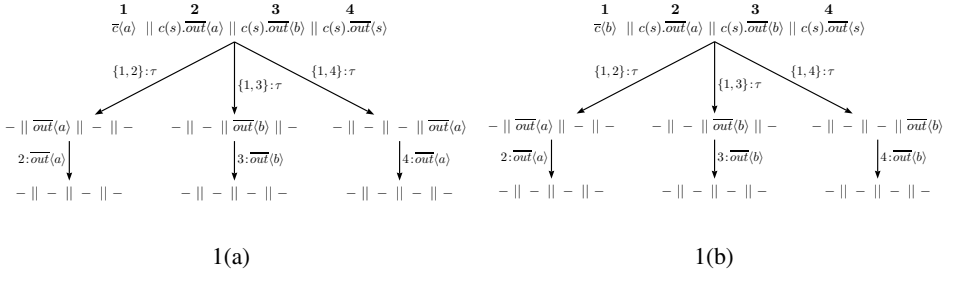

Fig. 1. Automata $S\left[{}^{a}/_{sec}\right]$ and $S\left[{}^{b}/_{sec}\right]$

4 Admissible Schedulers

In this section we restrict the discerning power of the global and local schedulers in order to avoid the problem of the information leakage induced in security by clairvoyant schedulers. We impose two kinds of restrictions: For the global scheduler, following [3], we assume that it can only see, and keep memory of, the observable actions and

the components that are enabled, but not the secret actions. As for the local scheduler, we assume that the local nondeterminism of each component is solved on the basis of the view of the history local to that component, i.e. the projection of the history of the system on that component. In other words, each component has to make decisions based only on the history of its own execution; it cannot see anything of the other components.

4.1 Restricting Global Schedulers

We assume that the set of actions Σ is divided in two parts, the *secret actions* $\mathcal{S}$ and the *observable actions* $\mathcal{O}$. The secret actions are supposed to be invisible to the global scheduler. Formally, this can be achieved using a function $sift$ with $sift(a)$ equals τ if $a \in \mathcal{S}$ and equals a otherwise. Then, we restrict the power of the global scheduler by forcing it to make the same decisions on paths he cannot tell apart.

Definition 3. *Given a TPA M, a global scheduler ζ for M is admissible if for all paths σ_1 and σ_2 we have $t(\sigma_1) = t(\sigma_2)$ implies $\zeta(\sigma_1) = \zeta(\sigma_2)$, where* $t\left(\hat{q} \xrightarrow{l_1:a_1} q_1 \xrightarrow{l_2:a_2} \cdots \xrightarrow{l_n:a_n} q_{n+1}\right) \stackrel{\text{def}}{=} (enab(\hat{q}), sift(a_1), l_1)(enab(q_1), sift(a_2), l_2)\cdots(enab(q_n), sift(a_n), l_n)$.

The idea is that t sifts the information of the path that the scheduler can see. Since $sift$ "hides" the secrets, the scheduler cannot take different decisions based on secrets.

4.2 Restricting Local Schedulers

The restriction on the local scheduler is based on the idea that a step of the component i of a system can only be based on the view that i has of the history, i.e. its own history. In order to formalize this restriction, it is convenient to introduce the concept of i-view of a path σ, or *projection* of σ on i, which we will denote by $\sigma_{\restriction i}$. We define it inductively:

$$(\sigma \xrightarrow{\ell:a} \mu)_{\restriction i} = \begin{cases} \sigma_{\restriction i} \xrightarrow{i:b} \delta_{q_i} & \text{if } \ell = \{i, j\} \text{ and } \mu = \delta_{(A)\ q_1 \| \ldots \| q_i \| \ldots \| q_j \| \ldots \| q_n} \\ \sigma_{\restriction i} \xrightarrow{i:a} \mu & \text{if } \ell = i \\ \sigma_{\restriction i} & \text{otherwise} \end{cases}$$

In the above definition, the first line represents the case of a synchronization step involving the component i, where we assume that the premise for i is of the form $q_i' \xrightarrow{b} \delta_{q_i}$. The second line represents an interleaving step in which i is the active component. The third line represents step in which the component i is idle.

The restriction to the local scheduler can now be expressed as follows:

Definition 4. *Given a TPA M and a local scheduler ξ for M, we say that ξ is admissible if for all paths σ and σ', if $\xi(\sigma) = (\ell, a, \mu)$, and $\xi(\sigma') = (\ell', a', \mu')$ we have:*

- *if $\ell = \ell' = i$ and $\sigma_{\restriction i} = \sigma'_{\restriction i}$, then $\xi(\sigma) = \xi(\sigma')$,*
- *if $\ell = \ell' = \{i, j\}$, $\sigma_{\restriction i} = \sigma'_{\restriction i}$, and $\sigma_{\restriction j} = \sigma'_{\restriction j}$ then $\xi(\sigma) = \xi(\sigma')$.*

A pair of compatible schedulers (ζ, ξ) is called *admissible* if ζ and ξ are admissible.

5 Safe Equivalences

In this section we revise process equivalence notions to make them safe for security.

5.1 Safe Complete Traces

We define here a safe version of complete-trace semantics. The idea is that we compare two processes based not only on their traces, but also on the choices that the global scheduler makes at every step. We do this by recording explicitly the tags in the traces.

Definition 5

- *Given a TPA $M = (Q, L, \Sigma, \hat{q}, \alpha)$, the (complete) safe traces of M, denoted here by $Traces_s$, are defined as the probabilities of sequences of tags and actions corresponding to all possible complete executions, i.e.*

$$Traces_s(M) = \{f : (L \times \Sigma)^\infty \to [0,1] \mid \text{there exists an admissible scheduler } (\zeta, \xi) \text{ s.t.} \forall t \in (L \times \Sigma)^\infty \; f(t) = \mathbf{P}_{M,\zeta,\xi}(\{\sigma \in \mathrm{CPaths}(M) \mid trace_s(\sigma) = t\}) \}$$

  where $\mathbf{P}_{M,\zeta,\xi}$ is the probability measure in M under (ζ, ξ), and $trace_{ta}$ extracts from a path the sequence of tags and actions, i.e. $trace_{ta}(\epsilon) = \epsilon$ (on the empty path $trace_{ta}$ gives the empty string) and $trace_{ta}(q \xrightarrow{\ell:a} \sigma) = \ell : a \cdot trace_{ta}(\sigma)$.
- *We denote by $Traces_s(q)$ the safe traces of the automaton associated to a system q.*
- *Two systems q_1 and q_2 are safe-trace equivalent, denoted by $q_1 \simeq_s q_2$, if and only if $Traces_s(q_1) = Traces_s(q_2)$.*

The following example points out the difference between $\simeq_s$ and the standard (complete) trace equivalence.

Example 3. Consider the TPAs of Example 2. The two TPAs have the same complete traces. In fact $Traces(S\,[^a/_{sec}]) = \{\tau \cdot \overline{out}\langle a\rangle\ ,\ \tau \cdot \overline{out}\langle b\rangle\} = Traces(S\,[^b/_{sec}])$. On the other hand, we have $Traces_s(S\,[^a/_{sec}]) = \{f_1, f_2, f_3\}$ where $f_1(\{1,2\} : \tau \cdot 2 : \overline{out}\langle a\rangle) = f_2(\{1,3\} : \tau \cdot 3 : \overline{out}\langle b\rangle) = f_3(\{1,4\} : \tau \cdot 4 : \overline{out}\langle a\rangle\}) = 1$, and $f_i(t) = 0$ otherwise (for $i \in \{1,2,3\}$), while $Traces_s(S\,[^b/_{sec}]) = \{f_1, f_2, f_4\}$ with f_1, f_2 as above, and $f_4(\{1,4\} : \tau \cdot 4 : \overline{out}\langle b\rangle) = 1$, $f_4(t) = 0$ otherwise.

5.2 Safe Bisimilarity

In this section we propose a security-safe version of strong bisimulation, that we call *safe bisimulation*. This is an equivalence relation stricter than safe-trace equivalence, with the advantage of being a congruence. Since in this paper schedulers can always observe which component is making a step (even a silent step), it does not seem natural to consider weak bisimulation.

We start with some notation. Given a TPA $M = (Q, L, \Sigma, \hat{q}, \alpha)$, and a global scheduler ζ, we write $q \xrightarrow{a}_{\zeta} \mu$ if there exists $\sigma \in \mathrm{Paths}^{\star}(M)$ such that $\zeta(\sigma) \neq \bot$, $(\zeta(\sigma), a, \mu) \in \alpha(q)$, and $q = last(\sigma)$. Note that the restriction to ζ still allows nondeterminism, i.e. there may be μ_1, μ_2, such that $q \xrightarrow{a_1}_{\zeta} \mu_1$ and $q \xrightarrow{a_2}_{\zeta} \mu_2$ (with either $a_1 = a_2$ or $a_1 \neq a_2$).

We now define the notion of safe bisimulation. The idea is that, if q and q' are bisimilar states, then every move from q should be mimicked by a move from q' *using the same (admissible) scheduler*.

Definition 6. *Given a TPA $M = (Q, L, \Sigma, \hat{q}, \alpha)$, we say that a relation $\mathcal{R} \subseteq Q \times Q$ is a safe bisimulation if, whenever $q_1 \mathcal{R} q_2$, then $enab(q_1) = enab(q_2)$, and for all admissible global schedulers ζ for M such that $\zeta(\sigma_1) = \zeta(\sigma_2)$ whenever $last(\sigma_1) = q_1$ and $last(\sigma_2) = q_2$:*

- *if $q_1 \xrightarrow{a}_{\zeta} \mu_1$, then there exists μ_2 such that $q_2 \xrightarrow{a}_{\zeta} \mu_2$ and $\mu_1 \mathcal{R} \mu_2$, and*
- *if $q_2 \xrightarrow{a}_{\zeta} \mu_2$, then there exists μ_1 such that $q_1 \xrightarrow{a}_{\zeta} \mu_1$ and $\mu_1 \mathcal{R} \mu_2$,*

where $\mu_1 \mathcal{R} \mu_2$ means that for all equivalence classes $X \in Q_{\hat{\mathcal{R}}}$, we have $\mu_1(X) = \mu_2(X)$, where $\hat{\mathcal{R}}$ is the smallest equivalence class induced by $\mathcal{R}$.

The following result is analogous to the case of standard bisimulation:

Proposition 1. *The union of all the safe bisimulations is still a safe bisimulation.*

Therefore the largest safe bisimulation exists, and coincides with the union of all safe bisimulations. We call it *safe bisimilarity*, and we denote it by $\sim_s$.

Given two TPAs on the same L and Σ, $M_1 = (Q_1, L, \Sigma, \hat{q}_1, \alpha_1)$ and $M_2 = (Q_2, L, \Sigma, \hat{q}_2, \alpha_2)$, we can define bisimulation and bisimilarity across their states, i.e. as relations on $(Q_1 \cup Q_2)$, in the obvious way, by constructing the TPA M with a new initial state $\hat{q}$ and two transitions to $\delta_{\hat{q}_1}$ and to $\delta_{\hat{q}_2}$, respectively.

Given two components or systems, q_1 and q_2, we will say that q_1 and q_2 are safely bisimilar, denoted by $q_1 \sim_s q_2$, if the initial states of the corresponding TPAs are safely bisimilar. Note that $q_1 \sim_s q_2$ is possible only if q_1 and q_2 have the same number of active components, where "active", for a component, means that during the execution of the system it will make at least one step. Note that in the case of components, or of systems constituted by one component only, safe bisimulation and safe bisimilarity coincide with standard bisimulation and bisimilarity (denoted by $\sim$), respectively. This is not the case for systems, as shown by the following example:

Example 4. Consider again the TPAs of Example 2. As pointed out in the introduction, we have $S\left[{}^{a}/_{sec}\right] \sim S\left[{}^{b}/_{sec}\right]$. However $S\left[{}^{a}/_{sec}\right] \not\sim_s S\left[{}^{b}/_{sec}\right]$. To show this, let us construct a new TPA (as described before) with initial state $\hat{q}$ such that $\hat{q} \xrightarrow{\ell:\tau} S\left[{}^{a}/_{sec}\right]$ and $\hat{q} \xrightarrow{\ell:\tau} S\left[{}^{b}/_{sec}\right]$. Now consider the (admissible) global scheduler ζ such that

$$\zeta(\sigma) \stackrel{\text{def}}{=} \begin{cases} \ell & if\ \sigma = \hat{q}, \\ \{1,4\} & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left[{}^{a}/_{sec}\right], \\ 2 & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left[{}^{a}/_{sec}\right] \xrightarrow{1,2:\tau} (\ - \,||\, \overline{out}\langle a\rangle \,||\, - \,||\, -\ ), \\ 3 & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left[{}^{a}/_{sec}\right] \xrightarrow{1,3:\tau} (\ - \,||\, - \,||\, \overline{out}\langle b\rangle \,||\, -\ ), \\ 4 & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left[{}^{a}/_{sec}\right] \xrightarrow{1,4:\tau} (\ - \,||\, - \,||\, - \,||\, \overline{out}\langle a\rangle\ ), \\ \{1,4\} & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left[{}^{b}/_{sec}\right], \\ 2 & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left[{}^{b}/_{sec}\right] \xrightarrow{1,2:\tau} (\ - \,||\, \overline{out}\langle a\rangle \,||\, - \,||\, -\ ), \\ 3 & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left[{}^{b}/_{sec}\right] \xrightarrow{1,3:\tau} (\ - \,||\, - \,||\, \overline{out}\langle b\rangle \,||\, -\ ), \\ 4 & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left[{}^{b}/_{sec}\right] \xrightarrow{1,4:\tau} (\ - \,||\, - \,||\, - \,||\, \overline{out}\langle b\rangle\ ), \\ \bot & otherwise. \end{cases}$$

It is easy to see that $S\left[{}^{b}/_{sec}\right]$ cannot mimic the transition $4 : \overline{out}\langle a\rangle$ produced by $S\left[{}^{a}/_{sec}\right]$ using the same scheduler ζ.

It turns out that safe bisimulation is a congruence with respect to all the operators of our language, as expressed by the following theorem. (Statements $2(a)$ and $2(b)$ are just the standard compositionality result for probabilistic bisimulation.)

Theorem 1

1. *$\sim_s$ is an equivalence relation.*
2. *Let $a \in \Sigma$ and $A, B, B' \subseteq \Sigma$. Let $p_1, \ldots, p_n$ be probability values, and let $q, q_1, q_2, \ldots, q_n, q'_1, q'_2, \ldots, q'_n$ be components.*
   (a) If $q_1 \sim_s q_2$, then $a.q_1 \sim_s a.q_2$, $q_1 + q \sim_s q_2 + q$, $(a)q_1 \sim_s (a)q_2$, and $q_1 \mid q \sim_s q_2 \mid q$.
   (b) If $q_1 \sim_s q'_1, \ldots, q_n \sim_s q'_n$, then $\sum_i p_i : q_i \sim_s \sum_i p_i : q'_i$.
   (c) If $(B)\ q_1 \parallel \ldots \parallel q_n \sim_s (B')\ q'_1 \parallel \ldots \parallel q'_n$, and $fn(q) \notin B \cup B'$, then

$$(A \cup B)\ q_1 \parallel \ldots \parallel q \parallel \ldots \parallel q_n \sim_s (A \cup B')\ q'_1 \parallel \ldots \parallel q \parallel \ldots \parallel q'_n.$$

The following property shows that bisimulation is stronger than safe-trace equivalence, like in the standard case.

Proposition 2. *If $q_1 \sim_s q_2$ then $q_1 \simeq_s q_2$.*

Like in the standard case, the vice-versa does not hold, and safe-trace equivalence is not a congruence[3].

6 Safe Nondeterministic Information Hiding

In this section we define the notion of information hiding under the most general hypothesis that the nondeterminism is handled partly in a demonic way and partly in an

[3] This is because we are considering the *complete* traces.

angelic way. We assume that the demonic part is in the realm of the global scheduler, while the angelic part is controlled by the local scheduler. The motivation is that in a protocol the local components can be thought of as programs running locally in a single machine, and locally predictable and controllable, while the network can be subject to attacks that make the interactions unpredictable.

We recall that, in a purely probabilistic setting, the absence of leakage, such as no-interference and strong anonymity, is expressed as follows (see for instance [5]). Given a purely probabilistic automaton M, and a sequence $\tilde{a} = a_1 a_2 \dots a_n$, let $\mathbf{P}_M([\tilde{a}])$ represent the probability measure of all complete paths with trace $\tilde{a}$ in M. Let S be a protocol containing a variable action $secr$, and let s be secret actions. Let M_s be the automaton corresponding to $S[^s/_{secr}]$. Define $Pr(\tilde{a} \mid s)$ as $\mathbf{P}_{M_s}([\tilde{a}])$. Then S is leakage-free if for every observable trace $\tilde{a}$, and for every secret s_1 and s_2, we have $Pr(\tilde{a} \mid s_1) = Pr(\tilde{a} \mid s_2)$.

In a purely nondeterministic setting, on the other hand, the absence of leakage has been characterized in the literature by the property $S[^{s_1}/_{secr}] \cong S[^{s_2}/_{secr}]$, where $\cong$ is an equivalence relation like trace equivalence, or bisimulation. As we have argued in the introduction, this definition assumes an angelic interpretation of nondeterminism.

We want to combine the above notions so to cope with both probability and nondeterminism. Furthermore, we want to extend it to the case in which part of the nondeterminism is interpreted demonically. Let us first introduce some notation.

Let S be a system containing a variable action $secr$. Let s be a secret action. Let M_s be the TPA associated to $S[^s/_{secr}]$ and let (ζ, ξ) be a compatible pair of global and local schedulers for M_s. The probability of an observable trace $\tilde{a}$ given s is defined as $Pr_{\zeta,\xi}(\tilde{a} \mid s) = \mathbf{P}_{M_s,\zeta,\xi}([\tilde{a}])$.

The global nondeterminism is interpreted demonically, and therefore we need to ensure that the conditional of an observable, given the two secrets, are calculated with respect to the same global scheduler. On the other hand, the local scheduler is interpreted angelically, and therefore we can compare the conditional probabilities generated by the two secrets as sets under different schedulers. In other words, we have the freedom to match conditional probability from the first set with one of the other set, without requiring the local scheduler to be the same.

Either angelic or demonic, we want to avoid the clairvoyant schedulers, i.e. a scheduler should not be able to use the secret information to achieve its goals. For this purpose, we require both the global and the local scheduler to be admissible.

Definition 7. *A system is leakage-free if, for every secrets s_1 and s_2, every admissible global scheduler ζ, and every observable trace $\tilde{a}$, $\{Pr_{\zeta,\xi}(\tilde{a} \mid s_1) \mid \xi$ admissible and compatible with $\zeta\} = \{Pr_{\zeta,\xi}(\tilde{a} \mid s_2) \mid \xi$ admissible and compatible with $\zeta\}$.*

The safe equivalences defined in Section 5 imply the absence of leakage:

Theorem 2. *Let S be a system with a variable action $secr$ and assume $S[^{s_1}/_{secr}] \simeq_s S[^{s_2}/_{secr}]$ for every pair of secrets s_1 and s_2. Then S is leakage-free.*

Note that the vice versa is not true, i.e. it is not the case that the leakage-freedom of S implies $S[^{s_1}/_{secr}] \simeq_s S[^{s_2}/_{secr}]$. This is because in the definition of safe-trace equivalence we compare the set of probability functions (determined by the schedulers) on

traces, while in the definition of leakage-freedom we compare the set of probabilities of each trace, which may come from different functions. This additional degree of freedom generated by the local scheduler helps the system to obfuscate the secret, and provides further justification for the adjective "angelic" for the local nondeterminism.

From the above theorem and from Proposition 2, we also have the following corollary (with the same premises as the previous theorem):

Corollary 1. *If* $S[^{s_1}/_{secr}] \sim_s S[^{s_2}/_{secr}]$ *for every pair of secrets* s_1 *and* s_2*, then* S *is leakage-free.*

7 Related Work

The problem of deriving correct implementations from secrecy specifications has received a lot of attention already. One of the first works to address the problem was [18], which showed that the fact that an implementation is a consistent refinement w.r.t. a specification does not imply that the (information-flow) security properties are preserved. More recently, [2] has proposed a notion of secrecy-preserving refinement, and a simulation-based technique for proving that a system is the refinement of another. [11] argues that important classes of security policies such as noninterference and average response time cannot be expressed by traditional notion of *properties*, which consist of sets of traces, and proposes to use *hyperproperties* (sets of properties) instead. [14] addresses the problem of supervisory control, i.e, given a critical system G that may leak confidential information, how to design a controller C so that the system $G|C$ dos not leak. An effective algorithm is presented to compute the most permissible controller such that the system is still opaque w.r.t. a secret.

Concerning angelic and demonic nondeterminism, there are various works which investigate their relation and possible combination. In [4] it is shown that angelic and demonic nondeterminism are dual. [19] uses multi-relations to express specifications involving both angelic and demonic nondeterminism. There are two kinds of agents, demonic and angelic ones, and there is the point of view of the internal system and the one of the external adversary. [22] considers the problem of refining specifications while preserving ignorance. While the focus is on the reduction of demonic nondeterminism of the specification, the hidden values are treated essentially in a angelic way.

The problem of the leakage caused by full-information schedulers has also been investigated in literature. [6] and [7] work in the framework of probabilistic automata and introduce a restriction on the scheduler to the purpose of making them suitable to applications in security protocols. Their approach is based on dividing the actions of each component of the system in equivalence classes (*tasks*). The order of execution of different tasks is decided in advance by a so-called *task scheduler*, which is history-independent and therefore much more restricted than our notion of global scheduler. [3] proposes a notion of system and admissible scheduler very similar to our notion of system and admissible global scheduler. The main difference is that in that work the components are deterministic and therefore there is no notion of local scheduler.

The work in [9,8] is similar to ours in spirit, but in a sense *dual* from a technical point of view. Instead of defining a restriction on the class of schedulers, they provide a way to specify that a choice is transparent to the scheduler. They achieve this by introducing labels in process terms, used to represent both the states of the execution tree and the next action or step to be scheduled. They make two states indistinguishable to schedulers, and hence the choice between them private, by associating to them the same label. We believe that every scheduler in our formalism can be expressed in theirs, too. In [8] they also consider the problem of defining a safe version of bisimulation for expressing security properties. They call it *demonic bisimulation*. The main difference with our work is that we consider a combination of angelic and demonic nondeterminism, and this affects also the definition of bisimulation. Similarly, our definition of leakage-freedom reflects this combination. In [8] the aspect of angelicity is not considered, although they may be able to simulate it with an appropriate labeling.

The fact that full-information schedulers are unrealistic has also been observed in fields other than security. First attempts used restricted schedulers in order to obtain rules for compositional reasoning [12]. The justification for those restricted schedulers is the same as for ours, namely, that not all information is available to all entities in the system. However that work considers a synchronous parallel composition, so the setting is rather different from ours. Later on, it was shown that model checking is unfeasible in its general form for the restricted schedulers in [12] (see [16] and, more recently, [15]). Despite of undecidability, not all results concerning such schedulers have been negative as, for instance, the technique of partial-order reduction can be improved by assuming that schedulers can only use partial information [17].

8 Conclusion and Future Work

We have observed that some definitions of security properties based on process equivalences may be too naive, in that they assume the scheduler to be angelic, and, worse yet, to achieve its angelic strategy by peeking at the secrets. We have presented a formalism allowing us to specify a demonic constituent of the scheduler, possibly in collusion with the attacker, and an angelic one, under the control of the system. We have also considered restrictions on the schedulers to limit the power of what they can see, and extended to our nondeterministic framework the (probabilistic) information-hiding properties like non interference and strong anonymity. We then have defined "safe" equivalences. In particular we have defined the notions of safe trace equivalence and safe bisimilarity, and we have shown that the latter is still a congruence. Finally, we have shown that the safe equivalences can be used to prove information-hiding properties.

For the future, we plan to extend our framework to quantitative notions of information leakage, possibly based on information theory. We also plan to implement model checking techniques to verify information hiding properties for our kind of systems. A natural candidate for the implementation would be PRISM. Of course, we would need to restrict the class of schedulers in PRISM so to meet the admissibility criteria.

Acknowledgement. The authors wish to thank the anonymous reviewers for their useful comments, and Pedro D'Argenio for helpful discussion.

References

1. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The spi calculus. Inf. and Comp. 148(1), 1–70 (1999)
2. Alur, R., Zdancewic, S.: Preserving secrecy under refinement. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 107–118. Springer, Heidelberg (2006)
3. Andrés, M.E., Palamidessi, C., van Rossum, P., Sokolova, A.: Information hiding in probabilistic concurrent systems, `http://www.cs.ru.nl/M.Andres/downloads/SAuN.pdf`
4. Back, R.J.R., von Wright, J.: Combining angels, demons and miracles in program specifications. TCS 100(2), 365–383 (1992)
5. Bhargava, M., Palamidessi, C.: Probabilistic anonymity. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 171–185. Springer, Heidelberg (2005)
6. Canetti, R., Cheung, L., Kaynar, D., Liskov, M., Lynch, N., Pereira, O., Segala, R.: Task-structured probabilistic i/o automata. In: Proc. of WODES (2006)
7. Canetti, R., Cheung, L., Kaynar, D.K., Liskov, M., Lynch, N.A., Pereira, O., Segala, R.: Time-bounded task-PIOAs: A framework for analyzing security protocols. In: Dolev, S. (ed.) DISC 2006. LNCS, vol. 4167, pp. 238–253. Springer, Heidelberg (2006)
8. Chatzikokolakis, K., Norman, G., Parker, D.: Bisimulation for demonic schedulers. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 318–332. Springer, Heidelberg (2009)
9. Chatzikokolakis, K., Palamidessi, C.: Making random choices invisible to the scheduler. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 42–58. Springer, Heidelberg (2007)
10. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology 1, 65–75 (1988)
11. Clarkson, M.R., Schneider, F.B.: Hyperproperties. In: CSF, pp. 51–65. IEEE, Los Alamitos (2008)
12. de Alfaro, L., Henzinger, T.A., Jhala, R.: Compositional methods for probabilistic systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, p. 351. Springer, Heidelberg (2001)
13. Delaune, S., Kremer, S., Ryan, M.: Verifying privacy-type properties of electronic voting protocols. Journal of Computer Security 17(4), 435–487 (2009)
14. Dubreil, J., Darondeau, P., Marchand, H.: Supervisory control for opacity. IEEE Transactions on Automatic Control 55(5), 1089–1100 (2010)
15. Giro, S.: Undecidability results for distributed probabilistic systems. In: Oliveira, M.V.M., Woodcock, J. (eds.) SBMF 2009. LNCS, vol. 5902, pp. 220–235. Springer, Heidelberg (2009)
16. Giro, S., D'Argenio, P.R.: Quantitative model checking revisited: Neither decidable nor approximable. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 179–194. Springer, Heidelberg (2007)
17. Giro, S., D'Argenio, P.R., Fioriti, L.M.F.: Partial order reduction for probabilistic systems: A revision for distributed schedulers. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 338–353. Springer, Heidelberg (2009)
18. Jacob, J.: On the derivation of secure components. In: S&P, pp. 242–247. IEEE, Los Alamitos (1989)
19. Martin, C.E., Curtis, S.A., Rewitzky, I.: Modelling angelic and demonic nondeterminism with multirelations. Science of Computer Programming 65(2), 140–158 (2007)
20. Milner, R.: Communication and Concurrency. Series in Comp. Sci. Prentice Hall, Englewood Cliffs (1989)

21. Milner, R.: Communicating and mobile systems: the π-calculus. CUP (1999)
22. Morgan, C.: The shadow knows: Refinement and security in sequential programs. Science of Computer Programming 74(8), 629–653 (2009)
23. Reiter, M.K., Rubin, A.D.: Crowds: anonymity for Web transactions. ACM Transactions on Information and System Security 1(1), 66–92 (1998)
24. Schneider, S., Sidiropoulos, A.: CSP and anonymity. In: Martella, G., Kurth, H., Montolivo, E., Bertino, E. (eds.) ESORICS 1996. LNCS, vol. 1146, pp. 198–218. Springer, Heidelberg (1996)
25. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis, Tech. Rep. MIT/LCS/TR-676 (1995)
26. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing 2(2), 250–273 (1995)