Tempus Fugit: How To Plug It

Alessandra Di Pierro [1]

Dipartimento di Informatica, Universitá di Pisa, Italy

Chris Hankin [1] Igor Siveroni [1] Herbert Wiklicky [1]

Department of Computing, Imperial College London, United Kingdom

Abstract

Secret or private information may be leaked to an external attacker through the timing behaviour of the system running the untrusted code. After introducing a formalisation of this situation in terms of a confinement property, we present an algorithm which is able to transform the system into one that is computationally equivalent to the given system but free of timing leaks.

Key words: Security Analysis, Timing Leaks, Program Transformation, Probabilistic Processes

1 Introduction

This paper concerns the transformation of programs to remove covert channels. Our focus is on timing channels, which are able to signal information through the time at which an action occurs (a command is executed).

A system is vulnerable to a timing attack if there are alternative control flow paths of observably different duration; a common refinement of this definition would be just to focus on those alternative paths where the selection is guarded by confidential information. We will show that the comparison of different paths is essentially a confinement problem. We have studied confinement in [1] using probabilistic bisimulation. Specifically, we showed that two processes are probabilistically confined if and only if they share a common (probabilistic) abstract interpretation (Proposition 39 of [1]). Our approach to probabilistic abstract interpretation is based on Operator Algebra and thus

[1] All authors are partly funded by the EPSRC project S77066A.

abstractions are measurable by a norm; in particular, in *op. cit.*, we showed how to measure the difference between two processes and thus introduced the notion of approximate confinement. In *op. cit.* this measure, ϵ, was given a statistical interpretation which indicated how much work an attacker would have to perform in order to differentiate the processes.

In this paper we use our previous work to identify the need for transformation. We identify the alternative paths which fail to be bisimilar. These are then candidates for transformation. We present a transformation which establishes probabilistic bisimilarity whilst retaining input/output equivalence with the original system. This transformation involves the introduction of new "padded" control paths. This is similar to the approach of Agat [2]. However, in contrast to Agat, our work is not programming language specific; instead it is based on probabilistic transition systems which provides a very general setting. The transformed system produced by our approach is more efficient than that which would be produced by Agat's algorithm. This is because our algorithm introduces less padding steps.

In the next section, we review some background material on probabilistic transition systems and their representation as linear operators. Section 3 reviews the pertinent results from [1] and shows how to use these results to identify potential timing leaks. Section 4 and Section 5 present our algorithm for padding and its correctness. The remaining sections present a series of examples and comparisons to related work.

2 Probabilistic Processes

Probabilistic process algebras extend the well-established classical techniques for modelling and reasoning about concurrent processes with the ability of dealing with non-functional aspects of process behaviour such as performance and reliability, or in general the quantitative counterpart of classical properties representing the functional behaviour of a system.

Several models for probabilistic processes have been proposed in the literature in recent years, which differ from each other essentially in the "amount" of nondeterminism that is allowed in the model. Replacing nondeterministic branching by a probabilistic one leads to either the *reactive* or the *generative* model introduced in [3], where the probability distribution which guides the choice may depend (reactive) or not (generative) on the action labelling the branches (or transitions). Other models allow for both nondeterministic and probabilistic branching and lead to a myriad of variants depending on the interplay between the two kinds of branching (cf. [4]).

In order to describe the operational behaviour of probabilistic processes we consider *probabilistic transition systems* (PTS's), that is probabilistic extensions of the notion of labelled transition systems for classical process algebras. Probabilistic transition systems essentially consist of a set of states and a set of labels, and their behaviour can be informally described as follows: To a given action of the environment the system responds by either refusing it or making a transition to a new state according to a probability distribution. In this paper we consider only PTS's with a **finite** number of states and actions, which reflect the **generative model** of probability; they effectively correspond to a particular kind of probabilistic processes, namely finite Markov chains [5].

2.1 Probabilistic Transition Systems

Formally, we can define a probabilistic transition system in all generality as in [6]. We denote by $\mathit{Dist}(X)$ the set of all probability distributions on the set X, that is of all functions $\pi : X \to [0,1]$, such that $\sum_{x\in X} \pi(x) = 1$.

Definition 1 *A* probabilistic transition system *is a tuple* $(S, A, \longrightarrow, \pi_0)$*, where:*

- S *is a non-empty, finite set of* states*,*
- A *is a non-empty, finite set of* actions*,*
- $\longrightarrow \subseteq S \times A \times \mathit{Dist}(S)$ *is a* transition relation*, and*
- $\pi_0 \in \mathit{Dist}(S)$ *is an* initial distribution *on* S*.*

For $s \in S$, $\alpha \in A$ and $\pi \in \mathit{Dist}(S)$ we write $s \xrightarrow{\alpha} \pi$ for $(s, \alpha, \pi) \in \longrightarrow$. By $s \xrightarrow{p:\alpha} t$ we denote the transition to individual states t with probability $p = \pi(t)$. We denote the transition probability from a state s_1 to a state s_2 with label α by $p(s_1, \alpha, s_2)$. For a set of states C we write the *accumulated transition probability* from s_1 to C as: $p(s_1, \alpha, C) = \sum_{s\in C} p(s_1, \alpha, s)$.

In the *generative model* the transition relation of a PTS $(S, A, \longrightarrow, \pi_0)$ is a partial function $\longrightarrow: S \hookrightarrow \mathrm{Dist}(S \times A)$; this means that the same probability distribution is used to govern both the choice of the action and the (internal) state transition. Looking at a PTS as a Markov chain, we can identify it with the *tree* of the outcomes of the associated stochastic process; this can be seen as an experiment which takes place in stages. At each stage the probability for each possible outcome is known when the outcomes of all previous stages are known, and the number of the possible outcomes at each stage is finite. We can define the tree associated to a PTS inductively as follows.

Definition 2 *Let* S *is be a non-empty, finite set of states, and let* A *be a non-empty finite set of actions. Then*

- $(\{s\}, A, \emptyset, \{\langle s, 1\rangle\})$ *is a tree-PTS with root* $s \in S$*;*

- *if $T_i = (S_i, A, \longrightarrow_i, \{\langle s_i, 1\rangle\})$, $i = 1, \ldots, m$, $m < \infty$ are tree-PTS's with roots $s_i \in S_i$, $S_i \subseteq S$ for all $i = 1, \ldots, m$, $S_i \cap S_j = \emptyset$ for $i \neq j$, $s \in S \setminus \bigcup_i S_i$, and $\{\langle (s_i, \alpha_i), p_i\rangle \mid s_i \in S_i, \alpha_i \in A, i = 1, \ldots, m\}$ is a probability distribution, then the PTS constructed as $(\{s\} \cup \bigcup_i S_i, A, \{s \xrightarrow{p_i:\alpha_i} s_i\} \cup \bigcup \longrightarrow_i, \{\langle s, 1\rangle\})$ is a tree-PTS with root s. We call T_i the sub-trees of T.*

When the transition probabilities form a sub-probability distribution, that is a function $\pi : S \times A \to [0, 1]$ with $\sum_{s \in S} \pi(x) \leq 1$, then we call the PTS a subPTS (tree-subPTS).

2.2 Linear Representation of Probabilistic Transition Systems

As in previous work – e.g. [7,1] – we recast the common relational presentation of PTS's in terms of linear maps and operators in order to provide an appropriate mathematical framework for the quantitative study of probabilistic processes.

Definition 3 *Given a (sub)PTS $T = (S, A, \longrightarrow, \pi_0)$ and a fixed $\alpha \in A$ then the* matrix representing *the probabilistic transition relation $\xrightarrow{\alpha} \subseteq S \times [0, 1] \times S$ corresponding to α is given by*

$$(\mathbf{M}_{\xrightarrow{\alpha}})_{st} = \begin{cases} p \textit{ iff } s \xrightarrow{p:\alpha} t \\ 0 \textit{ otherwise} \end{cases}$$

where $s, t \in S$, and $(\mathbf{M})_{st}$ denotes the entry in column s and row t in the matrix $\mathbf{M}$. We define the matrix representation $\mathbf{M}(T)$ *of the (sub)PTS P as the direct sum of the matrix representations of the transition relations $\xrightarrow{\alpha}$ for each $\alpha \in A$:*

$$\mathbf{M}(T) = \bigoplus_{\alpha \in A} \mathbf{M}_{\xrightarrow{\alpha}}.$$

We recall that for a set $\{\mathbf{M}_i\}_{i=1}^k$ of $n_i \times m_i$ matrices, the *direct sum* of these matrices is defined by the $(\sum_{i=1}^k n_i) \times (\sum_{i=1}^k m_i)$ "diagonal" matrix:

$$\mathbf{M} = \bigoplus_i \mathbf{M}_i = \mathrm{diag}(\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_k)$$

Given k square, $n \times n$ matrices $\mathbf{M}_i$ and an $m \times n$ matrix $\mathbf{K}$ then we use the short hand notation $\mathbf{K}(\bigoplus_{i=1}^k \mathbf{M}_i) = (\bigoplus_{i=1}^k \mathbf{K})(\bigoplus_{i=1}^k \mathbf{M}_i)$, i.e. we apply $\mathbf{K}$ to each of the factors $\mathbf{M}_i$.

For probabilistic (sub-probabilistic) relations we obtain a so-called *stochastic* (*sub-stochastic*) matrix, i.e. a positive matrix where the entries in each row sum up to one (are less than or equal to one).

Matrices are not just schemes for writing down probabilities but also a way to specify *linear maps* or *operators* between/on vector spaces. In our case, PTS's are represented as operators on the *vector space* $\mathcal{V}(S) = \{ (v_s)_{s \in S} \mid v_x \in \mathbb{R}\}$, i.e. the set of formal linear combinations of elements in S with real coefficients. This space contains all distributions of the state space $Dist(S) \subset \mathcal{V}(S)$.

3 Process Equivalences and Confinement

The problem of preventing a system from leaking private information to unauthorised users is essentially the problem of guaranteeing the confinement [8] of the system against some specific attacks. Typically, after the introduction of the notion of noninterference [9,10], most of the work on confinement has been based on models which ultimately depend on some notion of process equivalence by identifying the absence of information flow between two processes via the indistinguishability of their behaviours [11].

In [1] it is shown how various process equivalences can be approximated, and how probabilities can be used as numerical information for quantifying such an approximation. This provides us with a quantitative measure of the indistinguishability of the process behaviour (according to the given semantics), that is in a security setting a measure of their propensity to leak information. In particular, a notion of confinement can be obtained by stating that two processes p and q are *probabilistically confined* iff they are probabilistic bisimilar.

The central aim of this paper is to present a transformation algorithm for PTS's which constructs bisimilar PTS's while preserving the computational effects of the original systems. This section is devoted to a formal definition of the two notions of process equivalence for PTS's at the base of our treatment, namely probabilistic bisimulation and probabilistic observables.

3.1 Probabilistic Bisimulation

Probabilistic bisimilarity is the standard generalisation of process bisimilarity to probabilistic transition systems and is due to [12, Def 4].

Definition 4 *A* probabilistic bisimulation *is an equivalence relation* $\sim_b$ *on* *the states of a probabilistic transition system satisfying for all $\alpha \in A$:*

$$p \sim_b q \text{ and } p \xrightarrow{\alpha} \pi \Rightarrow q \xrightarrow{\alpha} \varrho \text{ and } \pi \sim_b \varrho.$$

Note that in the case of generative systems considered in this paper, π and ϱ are sub-probability distributions.

The notion of probabilistic bisimulation on PTS corresponds to the notion of *lumped process* for finite Markov chains [5]. This can be seen by noting that a probabilistic bisimulation equivalence $\sim$ on a PTS $T = (S, A, \rightarrow, \pi_0)$ defines a probabilistic abstract interpretation of T [1]. In fact, we have shown that it is always possible to define a linear operator $\mathbf{K}$ from the space $\mathcal{V}(S)$ to the space $\mathcal{V}(S/_\sim)$ which represents $\sim$. This so-called classification operator is represented by the $n \times m$-matrix:

$$(\mathbf{K})_{ij} = \begin{cases} 1 \text{ if } s_i \in C_j \\ 0 \text{ otherwise} \end{cases}$$

with $S/_\sim = \{C_1, C_2, \ldots C_m\}$ and n the cardinality of S. If $\mathbf{M}(T)$ is the operator representation of T then $\mathbf{K}^\dagger\mathbf{M}(T)\mathbf{K}$ is the abstract operator induced by $\mathbf{K}$ where $\mathbf{K}^\dagger$ is the so-called *Moore-Penrose pseudo-inverse* of $\mathbf{K}$. For classification operators $\mathbf{K}$ we can construct $\mathbf{K}^\dagger$ as the row-normalised transpose of $\mathbf{K}$. Intuitively, $\mathbf{K}^\dagger\mathbf{M}(T)\mathbf{K}$ is an operator which abstracts the original system T by encoding only the transitions between equivalence classes instead of the ones between single states. $\mathbf{K}$ determines a partition on the state space of T which is exactly the lumping of the states of T. This can be formally deduced from the following theorem in [5].

Theorem 5 *A necessary and sufficient condition for a Markov chain to be lumpable with respect to a partition $\{C_1, C_2, \ldots C_m\}$ is that for every pair of sets C_i and C_j, the probability $p_{kC_j} = \sum_{t \in C_j} p_{it}$ of moving in one step from state s_k into set C_j have the same value for every $s_k \in C_i$. These common values $\{\tilde{p}_{ij}\}$ form the transition matrix for the lumped chain.*

We can re-formulate the probabilistic bisimilarity of two processes A and B in terms of linear operators as follows, cf [1]:

Proposition 6 *Given the operator representation $\mathbf{M}(A)$ and $\mathbf{M}(B)$ of two probabilistic processes A and B, then A and B are probabilistic bisimilar, that is $A \sim_b B$, iff there exist classification matrices $\mathbf{K}_A$ and $\mathbf{K}_B$ such that*

$$\mathbf{K}_A^\dagger\mathbf{M}(A)\mathbf{K}_A = \mathbf{K}_B^\dagger\mathbf{M}(B)\mathbf{K}_B.$$

Probabilistic bisimilarity of A and B can be expressed in terms of Markov chains lumpability as follows:

Proposition 7 *Given the operator representation* $\mathbf{M}(A)$ *and* $\mathbf{M}(B)$ *of two probabilistic processes* A *and* B*, then* A *and* B *are probabilistic bisimilar iff there exists a lumping* $\mathbf{K}$ *of the process* $\mathbf{M}(A) \oplus \mathbf{M}(B)$ *of the form*

$$\mathbf{K} = \begin{pmatrix} \mathbf{K}_A \\ \mathbf{K}_B \end{pmatrix}$$

for some classification operators $\mathbf{K}_A$ *and* $\mathbf{K}_B$ *for* A *and* B *such that* $\mathbf{K}_A$ *and* $\mathbf{K}_B$ *are fully column-ranked.*

For tree-PTS the above result essentially means that an equivalence relation on the union of the states of two processes A and B is a probabilistic bisimulation for A and B iff their roots belong to the same equivalence class.

Probabilistic bisimulation is the finest of all process equivalences as it identifies processes whose step-wise behaviour is exactly the same. From an external viewpoint other behaviours can be observed in order to distinguish two processes. In particular, a potential attacker may be able to obtain experimentally two kinds of information about a process by observing what final outcomes the process produces, and by measuring how long the process is running. Both the input/output behaviour and the timing behaviour of a process correspond to some suitable *abstractions* of the concrete behaviour of the process in question. In this paper we will concentrate on these observables and show how we can transform a process (PTS) in a way that its input/output behaviour is preserved, while its timing behaviour changes so as to become indistinguishable from another given process.

3.2 Probabilistic Observables

In our PTS model, any execution of a process leads to a single trace, i.e. a single execution path. Each time there is a probabilistic choice, one of the possible branches is chosen with some probability. The combined effect of all the choices made defines the probability of a particular path as the product of probabilities along this computational path. For finite processes we only have to consider finitely many paths of finite length and thus only need a basic probability model. This is based on a finite set of events Ω, i.e. all possible traces, such that all sets of traces have a well-defined probability attached, i.e. are measurable, and the probability function $\mathbf{P} : \mathcal{P}(\Omega) \rightarrow [0, 1]$ can be defined via a distribution, i.e. a function $\pi : \Omega \rightarrow [0, 1]$ such that $\mathbf{P}(X) = \sum_{x \in X} \pi(x)$

for all $X \subseteq \Omega$. This avoids the introduction of more complicated measure-theoretic structures on the set of possible paths.

The notion of observables we will consider is based on a *trace semantics* for probabilistic processes whose intuition and formal definition are given below.

3.2.1 Trace Semantics

Formally, the trace semantics of a process T is defined via the notion of probabilistic result of a computational path.

Definition 8 *For a computational path* $\pi = s_0 \xrightarrow{p_1:\alpha_1} s_1 \xrightarrow{p_2:\alpha_2} \dots \xrightarrow{p_n:\alpha_n} s_n$*, we define its* probabilistic result $\mathcal{R}(\pi)$ *as the pair* $\langle \alpha_1\alpha_2\dots\alpha_n, prob(\pi)\rangle$*, where* $prob(\pi) = \prod_i p_i$*. The trace associated to* π *is* $t(\pi) = \alpha_1\alpha_2\dots\alpha_n$*.*

Since different computational paths may have the same associated trace (cf. Example 9), we need to sum up the probabilities associated to each such computational path in order to appropriately define a probabilistic result. Formally, this can be done by defining for a process T and trace t the set

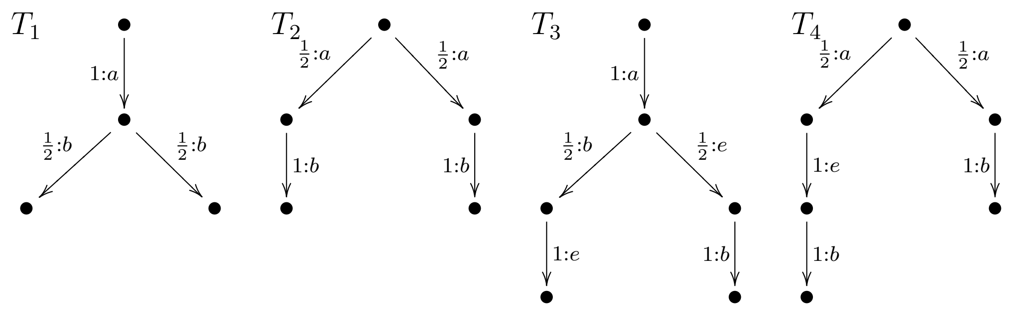
$$[\pi]_{T,t} = \{\pi \mid \pi \text{ a computational path for } T \text{ and } t(\pi) = t\},$$

which collects all the computational paths with trace t. Then we define for each trace t the computational result $\mathcal{R}([\pi]_{T,t})$ as the pair $\left\langle t, \sum_{\pi\in[\pi]_{T,t}} prob(\pi)\right\rangle$.

We can now define the trace semantics of T as the set

$$\mathcal{S}(T) = \{\mathcal{R}([\pi]_{T,t}) \mid t = t(\pi) \text{ and } \pi \text{ is a computational path for } T\}.$$

Example 9 *Consider the following simple processes with action* $A = \{a, b, e\}$*:*



It is easy to see that the possible traces for the first two processes are given by the sequence ab while for process T_3 *we have the traces abe and aeb and for process* T_4 *we have aeb and ab.*

We can thus describe the trace semantics of these four processes by:

$$\mathcal{S}(T_1) = \{\langle ab, 1\rangle\}$$
$$\mathcal{S}(T_2) = \{\langle ab, 1\rangle\}$$
$$\mathcal{S}(T_3) = \{\langle abe, 0.5\rangle, \langle aeb, 0.5\rangle\}$$
$$\mathcal{S}(T_4) = \{\langle aeb, 0.5\rangle, \langle ab, 0.5\rangle\}$$

3.2.2 I/O Observables

The notion of input/output observables, which we will refer to as I/O observables from now on, correspond to the most abstract observation of a process behaviour as it just considers the final result of an execution. In our setting, these final results are identified with *probability distributions* over the set of all possible *abstract traces*. Abstract traces are defined in terms of a function $[\![.]\!]$ which allows us to avoid specifying a concrete semantics for the labels, thus keeping our approach as general as possible. In fact, in a PTS we can think of labels as some kind of "instructions" which are preformed by the process, e.g. $a =$ "`x := x-1`", $b =$ "`x := 2*x`", etc. The final result of an execution is thus the accumulated effect of executing these instructions. Depending on the specific application, the function $[\![.]\!]$ will be defined so as to abstract from the appropriate computational details of the concrete traces.

For our purposes, we can leave the concrete nature of $[\![.]\!]$ unspecified and only assume that there is a neutral instruction e, i.e. a label which represents something like a `skip` statement, such that for any trace $t_1 t_2 \ldots t_n$ we have:

$$[\![t_1 t_2 \ldots t_n]\!] \equiv [\![t_1 t_2 \ldots t_i e t_{i+1} t_n]\!]$$

i.e. putting e anywhere in the execution (including at the end or the beginning) does not change the computational result. We will identify traces if the only difference between them are some occurrences of this neutral label e; this introduces an equivalence relation $\equiv$ on the set of all traces.

Clearly, different concrete traces might be identified by $[\![.]\!]$; for example, if $a =$ "`x := x+1`" and $b =$ "`x := x-1`" one would usually have $[\![ab]\!] = [\![e]\!] = [\![ba]\!]$.

Definition 10 *Given a tree-PTS T, we define its I/O observables as*

$$\mathcal{O}(T) = \{\mathcal{R}([\pi]_{T,[\![t]\!]}) \mid t = t(\pi) \textit{ and } \pi \textit{ is a computational path for } T\}.$$

Proposition 11 *For any tree-PTS T, its I/O observables $\mathcal{O}(T)$ define a probability distribution on the set of states S.*

PROOF. By induction on the tree structure of T.

- If $T = (\{s\}, A, \emptyset, \{\langle s, 1\rangle\})$ then $\mathcal{O}(T)$ is obviously a probability distribution on S.
- If $T = (\{s\} \cup \bigcup_i S_i, A, \{s \xrightarrow{p_i:\alpha_i} s_i\} \cup \bigcup \longrightarrow_i, \{\langle s, 1\rangle\})$, then by the inductive hypothesis we have that for all $i = 1, \ldots, m$ that $\mathcal{O}(T_i)$ is a probability distribution on S_i. Therefore, $\mathcal{O}(T) = \sum_{i=1}^{m} p_i \cdot \mathcal{O}(T_i)$ is a probability distribution on $\bigcup_i S_i$. $\square$

Based on this notion of I/O observables we can now define when two processes are I/O equivalent.

Definition 12 *Two PTS's A and B are I/O equivalent, denoted by $A \sim_{io} B$, if they define the same probability distribution on the set of the equivalence classes of abstract traces, i.e. iff*

$$\mathcal{O}(A) = \mathcal{O}(B).$$

Example 13 *If we consider again the four processes from Example 9, we get the following I/O observables:*

$$\begin{aligned}
\mathcal{O}(T_1) &= \{\langle [\![ab]\!], 1\rangle\}\\
\mathcal{O}(T_2) &= \{\langle [\![ab]\!], 1\rangle\}\\
\mathcal{O}(T_3) &= \{\langle [\![abe]\!], 0.5\rangle, \langle [\![aeb]\!], 0.5\rangle\} = \{\langle [\![ab]\!], 1\rangle\}\\
\mathcal{O}(T_4) &= \{\langle [\![aeb]\!], 0.5\rangle, \langle [\![ab]\!], 0.5\rangle\} = \{\langle [\![ab]\!], 1\rangle\}
\end{aligned}$$

i.e. all four processes are I/O equivalent.

3.3 Timing Behaviour

An external observer may have the ability to measure the time a process needs to perform a given task. In order to do so she just needs to observe "execution times", while the concrete nature of the performed instruction is irrelevant for her purposes. In our model, the timing behaviour of a process can therefore be defined by abstracting from labels (instructions) and assuming that there is a function $|.|$ from labels to real numbers which measures the time it takes to execute the instruction represented by the label.

One can think of various choices for the timing function. For example, one can assume (as in [13]) that there is only one special label t with $|t| = 1$ which indicates time progress while all other labels take no execution time; in this case the time flow is measured by the number of these special labels occurring in a trace. Alternatively, one can assign a different duration to each label. In our model we assume that all labels need a strictly positive time to be executed, i.e. $|\alpha| > 0$ for all labels, and that the execution time of all labels is the same. In particular, while in general one could have various neutral labels e with different executions time, e.g. $|e_0| = 0$, $|e_1| = 1$, $\dots$, we will assume that there is only one neutral element, denoted by $\surd$, which consumes exactly one time unit.

Formally we define the time semantics of our processes in terms of the notion of *time bisimulation*, that is a bisimulation relation on PTS's where transitions are labelled only by action $\surd$.

Definition 14 *Given a PTS $T = (S, A, \longrightarrow, \pi_0)$, we define its* depleted *or* ticked *version $\hat{T}$, as the PTS $(S', A', \longrightarrow', \pi'_0)$ with $S' = S$, $A' = \{\surd\}$, $\pi'_0 = \pi_0$, and $\longrightarrow'$ defined by*

$$(s, \surd, \pi) \in \longrightarrow' \ \ \textit{iff} \ (s, \alpha, \pi) \in \longrightarrow \ \ \textit{for some} \ \alpha \in A.$$

Time bisimilarity is defined as a probabilistic bisimilarity (cf. Definition 4) for depleted versions of PTS.

Definition 15 *Given two PTS's A and B, we say that A and B are time bisimilar, denoted by $A \sim_{tb} B$, iff there is a probabilistic bisimulation on the states of the ticked versions of A and B.*

3.4 Timing Leaks

Our model for timing leaks essentially follows the noninterference based approach to confinement recalled at the beginning of this section. In particular, we define timing attacks in terms of the ability of distinguishing the timing behaviour of two processes expressed via the notion of time bisimilarity.

In [1] we have used the same idea to model probabilistic covert channels. The notion of process equivalence used in that paper is an approximate version of probabilistic bisimilarity as the purpose there is to provide an an estimate of the confinement of the system, rather than just checking whether it is confined or not. We aim to extending this current work toward a similar direction (cf. Section 8). To this purpose, the linear representation of PTS, i.e. their correspondence with processes which are Markov chains, is essential; in fact, as shown in [1], it provides the necessary quantitative framework for defining numerical estimates.

Definition 16 *We say that two processes T_1 and T_2 are* confined against timing attacks *or* time confined *iff their associated PTS are time bisimilar.*

We can re-formulate this definition of timing leaks in terms of lumping as done in Proposition 6 for probabilistic bisimilarity. This will be the base for the definition of the transformation technique we will present in Section 4.

Definition 17 *Given the operator representation $\mathbf{M}(T) = \bigoplus_{\alpha \in A} \mathbf{M}_\alpha(T)$ of a probabilistic process T then the linear representation of its ticked version is given by:*

$$\widehat{\mathbf{M}}(T) = \sum_{\alpha \in A} \mathbf{M}_\alpha(T)$$

Definition 18 *We say that A and B are* time bisimilar*, $A \sim_{tb} B$, iff there exist classification matrices $\mathbf{K}_A$ and $\mathbf{K}_B$, such that*

$$\mathbf{K}_A^\dagger \widehat{\mathbf{M}}(A) \mathbf{K}_A = \mathbf{K}_B^\dagger \widehat{\mathbf{M}}(B) \mathbf{K}_B.$$

In the following we present a technique for closing possible covert timing channels by transforming the system into an equivalent one which is free from timing leaks. The transformation algorithm we present is applicable to finite tree-PTS's. It uses ideas from the optimal lumping algorithm in [14] and the padding techniques from [2].

Intuitively, given a system formed by processes A and B, the algorithm is based on the construction of a lumping for the Markov chain formed by the direct sum of the two Markov chains representing the ticked versions of A and B. By Proposition 7, the resulting lumped process will be a probabilistic bisimulation for A and B iff the root nodes of A and B belong to the same class (or they are essentially the same state in the lumped process). In this case no transformation is needed as the two processes are confined against timing attacks; otherwise the algorithm produces two new processes A' and B' which exhibit the same I/O behaviour as A and B but are also time confined. The following diagram shows the effect of the transformation.

$$\begin{tikzcd}
A \arrow[r, dash, "\not\sim_{tb}"] \arrow[d, dash, "\sim_{io}"'] & B \arrow[d, dash, "\sim_{io}"] \\
A' \arrow[r, dash, "\sim_{tb}"'] & B'
\end{tikzcd}$$

4 Transforming Probabilistic Transition Systems

Checking whether two systems are bisimilar or not, aka the bisimulation problem in the literature, is important not only in concurrency theory but also in many other fields such as verification and model checking, where bisimulation is used to minimise the states space, but also in computer security where many non-interference properties can be specified in terms of bisimulation equivalence [15,1]. Various algorithms have been proposed which solve this problem in linear time, the most important one being the algorithm proposed by Paige and Tarjan in [16]. This algorithm has been adapted in [17] for probabilistic processes; in particular, the Derisavi et al. algorithm constructs the optimal lumping quotient of a finite Markov chain in linear time, and can then be used to check the probabilistic bisimilarity of two PTS's.

In this paper we take a transformational approach: we not only check whether two probabilistic processes are bisimilar or not, but in the case where the processes are not bisimilar then we transform them into two bisimilar ones. The transformation we propose is inspired by the Derisavi et al. algorithm but does not aim to construct an "optimal" partition in the sense of [16]. It rather aims to add missing states and adjust the transition probabilities when it is needed to make two given processes bisimilar. It also preserves the probabilistic observable of the original processes.

The algorithm we present consists in essentially two intertwined procedures: one constructs a lumping partition for the PTS union of two tree-PTS's; the other "fixes" the partition constructed by the lumping procedure if some properties are violated which guarantee the partition to be a probabilistic bisimulation for the given processes. Fixing corresponds to adding some missing states (padding) and adjusting the transition probabilities in order to achieve bisimilarity. Moreover, the transformation is performed in a way that the original I/O semantics (i.e. probabilistic observables) of the processes is preserved. The overall procedure is repeated until a partition is constructed which effectively makes the two processes probabilistically bisimilar.

Before embarking on a more detailed description of our algorithm, we introduce some necessary notions and definitions. In the rest of this paper we focus on finite tree-structured PTS.

For a PTS $(S, A, \longrightarrow, \pi_0)$ we will consider partitions $P = \{B_1, \ldots, B_n\}$ of S; we will refer to the B_i's as the *blocks* of the partition P. Given a state $s \in S$ and a partition P we define the cumulative probability of s wrt $B \in P$ as $p(s, B) = \sum\{q \mid s \xrightarrow{q} t \text{ with } t \in B\}$. We further define the backward image for a block B and a probability q as $pre_{q,\longrightarrow}(B) = \{s \mid \exists B' \subseteq B : p(s, B') = q\}$; this corresponds to the set of all states from which B (i.e a state in B) can be

```
1: procedure LUMPING($T$)
2:     Assume: $T$ is a tree-PTS
3:     $P \leftarrow \{S\}$
4:     while $\neg$STABLE($P, T$) do
5:         choose $B \in P$
6:         $P \leftarrow$ SPLITTING($B, P$)
7:     end while
8: end procedure
```

Algorithm 1. Lumping Algorithm

reached with probability q. We will often omit the index $\longrightarrow$ when it is clear from the context.

For a tree-PTS T we will refer to HEIGHT(T) as the maximal length of a path in T from the root to a terminal state. The n *layer cut-off* of a tree-PTS T is defined inductively as (i) the set of terminal nodes for $n = 0$, and (ii) for $n = i + 1$ the set of nodes with only direct links (paths of length one) to nodes in the i layer cut-off. The procedure CUTOFF(T, n) returns the n layer cut-off of T which we also denote by $T|_n$. The complement of the n layer cut-off is sometimes called the n *layer top* and the top nodes in the n layer cut-off form the n *layer*.

Auxiliary Procedures. An important feature of the transformation of PTS's we present in the following is the creation of new states. These will be copies of existing states or of a distinguished additional "dummy state" d. We will denote copies of a given state s by $s', s'', s''', \ldots$. For copies of the dummy state we usually will omit the primes. For a tree-PTS $T = (S, A, \longrightarrow, \{\langle r, 1\rangle\})$ we define its copy as the tree-PTS $T' = (S', A, \longrightarrow', \{\langle r', 1\rangle\})$ with $S' = \{s' \mid s \in S\}$ and $s' \xrightarrow{p:\alpha}{}' t'$ iff $s \xrightarrow{p:\alpha} t$.

In the main algorithm we assume a routine COPYTREE(T,s) which returns a copy of the sub-tree of a tree-PTS T which is rooted in state s (where s is a state in T). We will also use a routine DEPLETETREE(T, s) which takes a tree-PTS T and returns $\check{T}$, i.e. a copy of T where all states are replaced by copies of the dummy state d and all transitions are labelled by $\surd$.

Another auxiliary procedure is LINKING(T_1, s, α, p, T_2) whose effect is to introduce a (copy) of a tree-PTS T_1 into an existing tree-PTS T_2 at state s; the linking transition will be labelled by α and the associated transition probability will be p. A related procedure called JOINING(T_1, s, p, T_2) introduces a (copy) of a tree-PTS T_1 into an existing tree-PTS T_2 at state s by identifying the root of T_1 with s and by multiplying initial transitions with probability p.

```
1: procedure MaxLumping($T_1, T_2$)
2:     Assume: $T_1 = (S_1, A, \longrightarrow, \{\langle r_1, 1\rangle\})$ is a tree structured PTS
3:     Assume: $T_2 = (S_2, A, \longrightarrow, \{\langle r_2, 1\rangle\})$ is a tree structured PTS
4:     $n \leftarrow 0$                                          ▷ Base Case
5:     $P \leftarrow \{S_1 \cup S_2\}$                          ▷ Partition
6:     $S \leftarrow \{S_1 \cup S_2\}$                          ▷ Splitters
7:     while $n \leq$ Height($T_1 \oplus T_2$) do
8:         $TT \leftarrow$ CutOff($T_1 \oplus T_2, n$)          ▷ $n$ layer cut off
9:         $P \leftarrow \{B \cap TT \mid B \in P\}$            ▷ Partition below current layer
10:        $S \leftarrow P \cup \{(S_1 \cup S_2) \setminus TT\}$   ▷ Splitters (below and top)
11:        while $S \neq \emptyset$ do
12:            choose $B \in S$, $S \leftarrow S \setminus B$   ▷ Choose a splitter
13:            $P \leftarrow$ Splitting($B, P$)                 ▷ Split (current layer)
14:            $C \leftarrow \{B \in P \mid B \cap S_1 = \emptyset \vee B \cap S_2 = \emptyset\}$   ▷ Critical class(es)
15:            if $C \neq \emptyset$ then
16:                return C                                   ▷ Found critical class(es)
17:            end if
18:        end while
19:        $n \leftarrow n + 1$                                 ▷ Go to next level
20:    end while
21: end procedure
```

Algorithm 2. Maximal Lumping Algorithm

Lumping. As already mentioned, our algorithm is inspired by the Paige-Tarjan algorithm for constructing bisimulation equivalences [16], i.e. *stable* partitions, for a given transition system. The same ideas can be used for probabilistic transition relations $\xrightarrow{p}$; in this case we say that a partition $P = \{B_1, \dots, B_n\}$ of a set of states is stable with respect to $\xrightarrow{p}$ iff for all blocks B_i and B_j in P and probability q we have that $pre_q(B_i) \cap B_j = \emptyset$ or $B_j \subseteq pre_q(B_i)$.

The construction of the lumping partition is done via the refinement of an initial partition which identifies all states in one class and proceeds as in the algorithm Lumping (cf Algorithm 1). Starting with the initial partition $P = \{S\}$ the algorithm incrementally refines it by a splitting procedure. For a block B_i the splitting procedure Splitting(B_i, P) refines the current partition P by replacing each block $B_j \in P$ with $pre_p(B_i) \cap B_j \neq \emptyset$ for some probability p by the new blocks $pre_p(B_i) \cap B_j$ and $B_j \setminus pre_p(B_i)$. The block B_i is called a *splitter*. We will also assume the definition of a procedure Stable(P, T) which returns *true* iff partition P is stable for PTS T.

In order to construct a ticked probabilistic bisimulation $\sim_{tb}$ for two tree-PTS's T_1 and T_2 we apply a slightly modified lumping procedure (cf Algorithm 2) to the ticked version $\hat{T}$ of the union PTS $T = T_1 \oplus T_2$. The procedure MaxLumping(T_1, T_2) takes two tree-PTS's $T_1 = (S_1, A, \longrightarrow, \{\langle r_1, 1\rangle\})$ and

```
1: procedure PADDING($T_1, T_2$)
2:     $C \leftarrow$MAXLUMPING($T_1, T_2$)                    ▷ identify critical classes
3:     repeat
4:         if $C = \{C_i\}_i$ with $C_i \cap S_1 = \emptyset$, $\forall C_i$ then
5:             $D \leftarrow (\{d\}, \{\surd\}, \emptyset, \{\langle d, 1\rangle\})$
6:             $T_1 \leftarrow$ LINKING($D, d, \surd, 1, T_1$)
7:         else if $C = \{C_i\}_i$ with $\forall C_i : C_i \cap S_2 = \emptyset$ then
8:             $D \leftarrow (\{d\}, \{\surd\}, \emptyset, \{\langle d, 1\rangle\})$
9:             $T_2 \leftarrow$ LINKING($D, d, \surd, 1, T_2$)
10:        else if $C = \{C_i\}_i$ with $C_j \subseteq S_1 \neq \emptyset \neq C_k \subseteq S_2$ then
11:            choose $s_1 \in C_k$
12:            choose $s_2 \in C_j$
13:            FIXIT($T_1, s_1, T_2, s_2$)
14:        end if
15:        $C \leftarrow$MAXLUMPING($T_1, T_2$)
16:    until $C = \emptyset$                                    ▷ i.e. $T_1 \sim_{tb} T_2$
17:    return $T_1, T_2$
18: end procedure
```

Algorithm 3. Padding Algorithm

$T_2 = (S_2, A, \longrightarrow, \{\langle r_2, 1\rangle\})$ and tries to construct a probabilistic bisimulation for $\hat{T}_1$ and $\hat{T}_2$ by lumping $\hat{T}$.

At each step the procedure constructs a partition $P = \{B_1, B_2, \ldots, B_m\}$ of the set $S_1 \cup S_2$ and identifies a set of *critical blocks* $C = \{C_1, \ldots C_k\} \subseteq P$. A critical block C_j is an equivalence class which has representatives in only one of the two PTS's, i.e. such that $C_j \cap S_1 = \emptyset$ or $C_j \cap S_2 = \emptyset$. The procedure works on"layers", i.e. at each iteration MAXLUMPING(T_1, T_2) partitions the nodes in layer n by choosing a splitter among the blocks in the n layer cut-off; it moves on to the next layer only when all such blocks have been considered.

Padding. The padding procedure identifies the critical blocks by calling the procedure MAXLUMPING(T_1, T_2) and then transforms the two processes T_1 and T_2 in order to "fix" the transitions to the critical blocks. The transformation is performed by the procedure FIXIT which we will describe later. In the special case where the critical blocks do not contain any representative of either of the two processes, say T_1, the fixing consists in padding T_1 with a depleted sub-tree formed by its root only which is a dummy state. This is linked to the root of T_1 via a ticked transition with probability 1 by the LINKING procedure. In the general (more symmetric) case where some of the critical blocks contain only states from one process and some other critical blocks only states from the other process the transformation is made by the FIXIT procedure.

```
1: procedure FixIt($T_1, s_1, T_2, s_2$)
2:     for all $B_i$ blocks do                         ▷ determine all possible changes
3:         Assume: $s_1 \xrightarrow{p_{1i}} B_i$ and $s_2 \xrightarrow{p_{2i}} B_i$
4:         $p'_i \leftarrow \min(p_{1i}, p_{2i})$
5:     end for
6:     Assume: $B_k$ is such that $p'_k = p_{2k}$
7:     Assume: $B_l$ is such that $p'_l = p_{1l}$
8:     $r_k \leftarrow p_{1k} - p'_k$
9:     $r_l \leftarrow p_{2l} - p'_l$
10:    choose $t_{1k} \in B_k \cap S_1$ and $T_{1k}$ sub-tree starting with $s_1 \xrightarrow{1} t_{1k}$
11:    choose $t_{2k} \in B_k \cap S_2$ and $T_{2k}$ sub-tree starting with $s_1 \xrightarrow{1} t_{2k}$
12:    choose $t_{1l} \in B_l \cap S_1$ and $T_{1l}$ sub-tree starting with $s_2 \xrightarrow{1} t_{1l}$
13:    choose $t_{2l} \in B_l \cap S_2$ and $T_{2l}$ sub-tree starting with $s_2 \xrightarrow{1} t_{2l}$
14:                                ▷ make maximal possible changes of probabilities
15:    Assume: $s_1 \xrightarrow{q_{1k}} t_{1k}$, $s_1 \xrightarrow{q_{1l}} t_{1l}$, $s_2 \xrightarrow{q_{2k}} t_{2k}$, and $s_2 \xrightarrow{q_{2l}} t_{2l}$
16:    $r_{1k} \leftarrow \min(q_{1k}, r_k)$
17:    $q_{1k} \leftarrow (q_{1k} - r_{1k})$
18:    $r_{2l} \leftarrow \min(q_{2l}, r_l)$
19:    $q_{2l} \leftarrow (q_{2l} - r_{2l})$
20:                            ▷ construct and link up "compensating" subtrees
21:    $X^1_{kl}, Y^1_{kl} \leftarrow$ Padding($T_{1k}, \hat{T}_{1l}$)
22:    $X^2_{kl}, Y^2_{kl} \leftarrow$ Padding($\hat{T}_{2k}, T_{2l}$)
23:    Joining($T_1, s_1, r_{1k}, X^1_{kl}$)
24:    Joining($T_2, s_2, r_{2l}, Y^2_{kl}$)
25: end procedure
```

Algorithm 4. Fixing Algorithm

When among the critical blocks of the current partition there is at least one block C_k which contains only representatives of T_1 and one block C_j which contains only representatives of T_2, the fixing is more elaborate and involves both processes. In this case there must exist states $s_1 \in C_k$ and $s_2 \in C_j$ with different probabilities of reaching some other blocks B_i of the partition. The FixIt procedure selects two blocks B_k and B_l in previous layer such that the probabilities p_{1k}, p_{1l} of the transitions from s_1 to B_k and B_l and the probabilities p_{2k}, p_{2l} of the transitions from s_2 to B_k and B_l are such that $p_{1k} > p_{2k}$ and $p_{1l} < p_{2l}$. This choice is always possible. In fact, the sum of all probabilities from s_1 and s_2 to other classes sum up to one, i.e. $\sum_i p_{1i} = 1 = \sum_i p_{2i}$ where i covers all classes B_i reachable from s_1 or s_2. As s_1 and s_2 are in different (critical) blocks the probabilities must differ for at least one class B_i, i.e. there exists at least one index k with $p_{1k} \neq p_{2k}$; thus we have that either $p_{1k} > p_{2k}$ or $p_{1k} < p_{2k}$. Without loss of generality, let $p_{1k} > p_{2k}$, i.e. we found the class B_k as required. We now have $\sum_{i \neq k} p_{1i} + p_{1k} = 1 = \sum_{i \neq k} p_{2i} + p_{2k}$ and thus $\sum_{i \neq k} p_{1i} < \sum_{i \neq k} p_{2i}$. As the summation is over the same number of probabilities, i.e. positive numbers, there must be at least one summand in

$\sum_{i \neq k} p_{2i}$ such that $p_{2l} > p_{1l}$.

After that, FixIt selects four states $t_{1k} \in B_k \cap S_1$, $t_{2k} \in B_k \cap S_2$, $t_{1l} \in B_l \cap S_1$, and $t_{2l} \in B_l \cap S_2$ with $s_1 \xrightarrow{q_{1k}} t_{1k}$, $s_1 \xrightarrow{q_{1l}} t_{1l}$, $s_2 \xrightarrow{q_{2k}} t_{2k}$, and $s_2 \xrightarrow{q_{2l}} t_{2l}$ and the corresponding sub-trees T_{1k}, T_{1l}, T_{2k} and T_{2l} which are rooted in t_{1k}, t_{1l}, t_{2k} and t_{2l}. Note that since B_k and B_l are in the CutOff$(n-1)$ they are not critical blocks, thus it is always possible to find such state. The probabilities q_{1l} and q_{2k} stay unchanged but q_{1k} and q_{2l} will be decreased by the maximal possible amount. Ideally, this decrement should be r_k and r_l, but if q_{1k} and q_{2l} are smaller than r_k and r_l we only reduce q_{1k} and q_{2l} to zero, i.e. effectively we remove the corresponding sub-trees.

The states s_1 and s_2 now have the same probabilities to reach the classes B_l and B_k, i.e. have become bisimilar. However, this transformations has two flaws: (i) the system is no longer a PTS as the probabilities from s_1 and s_2 no longer add up to one, (ii) the distribution on the terminal states reachable in T_{1k} and T_{2l} are reduced. In order to fix this we have to construct "compensating trees" which are bisimilar and which reproduce the missing probabilities for the outputs. These two trees are constructed via calling Padding on a copy of T_{1k} and a depleted version of T_{1l}, and a copy of T_{2l} and a depleted version of T_{2k} respectively. Finally, these "compensating trees" are linked to s_1 and s_2 with a $\surd$ label and the missing probabilities r_{1k} and r_{2l}.

This way we obtain again a proper tree-PTS with normalised probability distributions for s_1 and s_2. Furthermore, the"compensating trees" produce exactly the same output as T_{1k} and T_{2l}, thus the resulting PTS's have the same overall output as the original ones. Finally, by construction the compensating trees are bisimilar as T_{1k} and T_{2k} are bisimilar; the same holds for T_{1l} and T_{2l}.

5 Correctness of Padding

In order to show that Padding(T_1, T_2) indeed produces in PTS's T_1' and T_2' which are (i) I/O-equivalent, more precisely weak trace equivalent, to T_1 and T_2 respectively and (ii) tick-bisimilar, i.e. $T_1' \sim_{tb} T_2'$, we first describe the effect of various operations performed in Padding on the observables and (non)bisimilar states.

Given two states s_1 and s_2 in two tree-subPTS's T_1 and T_2 respectively and a bisimulation relation $\sim$ on the union $T = T_1 \oplus T_2$ then we define a new subPTS $T^{\min}_{s_1,s_2}$ as a *common minimisation of transition probabilities of* s_1 *and* s_2 by changing the transition probabilities from each of the two states to any class C_k of the bisimulation relation $\sim$ and all labels α as follows:

$$
\begin{aligned}
p'(s_1,\alpha,C_k) &= \min(p(s_1,\alpha,C_k),p(s_2,\alpha,C_k))\\
p'(s_2,\alpha,C_k) &= \min(p(s_1,\alpha,C_k),p(s_2,\alpha,C_k))\\
p'(s_i,\alpha,s_j) &= p(s_i,\alpha,s_j) \text{ for } i \notin \{1,2\}
\end{aligned}
$$

where p denotes the probabilities in T and p' the probabilities in $T^{\min}_{s_1,s_2}$. We denote by $T^{\min}_1$and $T^{\min}_2$ the new subPTS's corresponding to T_1 and T_2 respectively.

As we require a change of only the (accumulated) probabilities from each state to all classes it is possible to achieve this in various ways by adjusting the state-to-state probabilities in a number of ways, i.e. the common minimisation is not unique.

We also define a new relation $\sim'$ on $T^{\min}_{s_1,s_2}$ as follows $s_i \sim' s_j$ if $s_i \sim s_j$ and $\{i,j\} \cap \{1,2\} = \emptyset$ and $s_1 \sim' s_2$ and $s_2 \sim' s_1$.

Lemma 19 *Given a state s_1 in T_1 and a state s_2 in T_2 and a bisimulation relation $\sim$ on $T = T_1 \oplus T_2$, then the equivalence relation $\sim'$ on $T^{\min}_{s_1,s_2}$ is a bisimulation relation in which s_1 and s_2 are bisimilar.*

PROOF. This follows directly from the definition of $\sim'$ and p'. $\square$

Note that $\sim'$ is not necessarily the coarsest bisimulation relation or lumping on/of $T^{\min}_{s_1,s_2}$. It could happen that there exists already a class of states which are bisimilar to s_1 and s_2 after minimisation.

Consider two tree-subPTS's T and S and a state t in T. The *insertion of S into T at t with probability q and label λ* as the tree-subPTS $T \lhd^t_{q:\alpha} S$ with

$$
\begin{aligned}
p'(t_i,\alpha,t_j) &= p(t_i,\alpha,t_j) \text{ for } t_i,t_j \in T \text{ and all labels } \alpha\\
p'(s_k,\alpha,s_l) &= p(s_k,\alpha,s_l) \text{ for } s_i,s_j \in S \text{ and all labels } \alpha\\
p'(t,\lambda,s_0) &= q
\end{aligned}
$$

where s_0 is the root state of S and p denotes the probabilities in T and S respectively, and p' the probabilities in $T \lhd^t_{q:\alpha} S$.

Lemma 20 *Given two pairs of bisimilar tree-subPTS's $T_1 \sim T_2$ and $S_1 \sim S_2$ and two bisimilar states t_1 and t_2 in T_1 and T_2 respectively, then the common insertion of S_1 and S_2 into T_1 and T_2 at t_1 and t_2 with probability q and label λ result in bisimilar subPTS's, i.e.*

$$T_1 \lhd^{t_1}_{q:\lambda} S_1 \sim T_2 \lhd^{t_2}_{q:\lambda} S_2$$

PROOF. We define the bisimulation relation/lumping of $T_1 \lhd_p^{t_1} S_1$ and $T_2 \lhd_p^{t_2} S_2$ as follows: For all states in $S_1 \oplus S_2$ and $T_1 \setminus \{t_1\} \oplus T_2 \setminus \{t_2\}$ we use the same relation as before. For t_1 and t_2 we exploit the fact that they are bisimilar with respect to all classes in $T_1 \oplus T_2$ and with respect to the (potential) new state/class $\{s_{01}, s_{02}\}$ formed by the roots of S_1 and S_2 respectively we have by definition the same probability p. □

Given a tree-subPTS T then we define a *root extension* of T as a tree-subPTS $r \lhd_{1:\omega}^{t_0} T$ where we add an additional state r and define the transition probabilities as:

$$\begin{aligned} p'(t_i, \alpha, t_j) &= p(t_i, \alpha, t_j) \text{ for } t_i, t_j \in T \text{ and all labels } \alpha \\ p'(r, \omega, t_0) &= 1 \end{aligned}$$

where t_0 is the root state of T and p denotes the probabilities in T and p' the probabilities in $r \lhd_{1:\omega}^{t_0} T$. This can also be seen as a special kind of insertion which explains the notation.

Lemma 21 *Root extension does not change the I/O observables.*

PROOF. All the paths π_i from the root t_0 to the leaves in T are replaced by paths of the form $\omega\pi_i$ from r to the leaves in $T' = r \lhd_{1:\omega}^{t_0} T$, but as $\omega\pi_i \equiv \pi_i$ we have the same observables. □

Let T be a tree-subPTS and S a sub-tree-subPTS of T, then the *deletion S in T* is denote by $T \ntriangleleft S$, i.e. this is the subPTS on the states of $T \setminus S$ obtained by restriction of the transition relation to these states. The deletion of a sub-tree changes the observables as follows.

Lemma 22 *Given a tree-subPTS T and S a sub-tree-subPTS of T such that the observables of T and S are $\mathcal{O}(T)$ and $\mathcal{O}(S)$ and the (unique) path from the root of T to the root of S is π_0 then the observables of $T \ntriangleleft S$ are given by:*

$$\mathcal{O}(T \ntriangleleft S) = \mathcal{O}(T) - \pi_0 \cdot \mathcal{O}(S)$$

with "$\cdot$" denoting the pre-fixing*: $\pi_0 \cdot \{\langle \pi_i, p_i \rangle\}_i = \{\langle \pi_0\pi_i, p_i \rangle\}_i$.*

PROOF. By eliminating S we also eliminate the paths through S to leaves. As the observables are distributions over paths we simply have to eliminate the contribution of these paths through S from the (complete) observables of T. The contribution of these paths is not just their "S-stage" but has to be prefixed by the contribution until S, i.e. its root, is reached, this is given by the unique path p_0. $\square$

Insertion of sub-trees changes observables as follows.

Lemma 23 *Given two tree-subPTS T and S and a state t in T such that the observables of T and S are $\mathcal{O}(T)$ and $\mathcal{O}(S)$ and the (unique) path from the root of T to t is π_0 then the observables of the insertion of S into T with probability p with action α has the following observables:*

$$\mathcal{O}(T \lhd^t_{p:\alpha} S) = \mathcal{O}(T) + p\ \pi_0\alpha \cdot \mathcal{O}(S)$$

i.e. we have to add the "p-weighted" paths in $\pi_0\alpha \cdot \mathcal{O}(S)$ to $\mathcal{O}(T)$.

PROOF. Essentially the same argument as before: We obtain the same paths and thus observables as in T alone, but have to add the contribution of the paths going through S. The latter have to be additionally be prefixed by the path from the root of T to t and the entry action α from t to the root of S. $\square$

We can also describe the change of observables if we only partially delete the subtree S, if we change the *entry probability* into S from T, i.e. the probability with which the root of S can be reached from $T \setminus S$, from p to p'. We denote the resulting subPTS by $T \ntriangleleft^p_{p'} S$.

Lemma 24 *Given a tree-subPTS T and S a sub-tree-subPTS of T such that the observables of T and S are $\mathcal{O}(T)$ and $\mathcal{O}(S)$ and the (unique) path from the root of T to the root of S is π_0 then the observables of $T \ntriangleleft S$ are given by:*

$$\mathcal{O}(T \ntriangleleft^p_{p'} S) = \mathcal{O}(T) - (1 - p'/p)(\pi_0 \cdot \mathcal{O}(S)).$$

PROOF. The argument is essentially the same as previously, however each path reaching a leaf through S is not completely removed but only its impact on the observables is reduced. In T the probability is a product of the form $\prod_k p_k \cdot p \cdot \prod_l p_l$ where the p_k indicate the computational steps in π_0, the probability p is the original entry probability into S, and p_l are the probabilities in S. In $T \ntriangleleft^p_{p'} S$ the probability of this path is changed to: $\prod_k p_k \cdot p' \cdot \prod_l p_l$. If we therefore remove the contribution of these paths in the original observables and add the contribution in the new subPTS we obtain the desired result: $\mathcal{O}(T \ntriangleleft^p_{p'} S) = \mathcal{O}(T) - \pi_0 \cdot \mathcal{O}(S) + \frac{p'}{p}\pi_0 \cdot \mathcal{O}(S) = \mathcal{O}(T) - (1 - \frac{p'}{p})\pi_0 \cdot \mathcal{O}(S)$. $\square$

Proposition 25 *Given two tree-subPTS's T_1 and T_2 such that $T_1 \sim_{io} T_2$ and assume that the n-cutoffs are bisimilar, i.e. $T_1|_n \sim T_2|_n$, then the common* *minimisation of* $T_1|_{n+1} \oplus T_2|_{n+1}$ *with respect to two non-bisimilar states* $s_1 \in T_1$ *and* $s_2 \in T_2$ *in layer* $n+1$ *fulfils*

(1) $T_1^{\min}|_n \sim_{tb} T_2^{\min}|_n$
(2) the number of non-bisimilar states in $T_1|_{n+1} \oplus T_2|_{n+1}$ *is strictly reduced*
(3) the observables change as follows:

$$\begin{aligned}\mathcal{O}(T_1') &= \mathcal{O}(T_1) - (1 - \min(p_1, p_2)/p_1)(\pi_1 \cdot \mathcal{O}(s_{1\triangle}))\\ \mathcal{O}(T_2') &= \mathcal{O}(T_2) - (1 - \min(p_1, p_2)/p_2)(\pi_2 \cdot \mathcal{O}(s_{2\triangle}))\end{aligned}$$

where π_1 *and* π_2 *are the paths from the roots in* T_1 *and* T_2 *to* s_1 *and* s_2 *respectively and* $s_{1\triangle}$ *and* $s_{2\triangle}$ *denote the subtrees in* T_1 *and* T_2 *with roots* s_1 *and* s_2 *respectively.*

PROOF. The bisimulation class a state s belongs to are determined in a finite tree-subPTS only by it successors, i.e. $s_\triangle$. As we only change the transitions for states s_1 and s_2 in layer $n+1$ nothing changes for the states in the n-cutoffs; this shows (1). We do not introduce any new states in layer $n+1$ but we force s_1 and s_2 to be bisimilar, cf Lemma 19. Therefore the number of non-bisimilar states in layer $n+1$ is strictly reduced (by two) and as all states in the n-cutoffs are bisimilar by assumption we have shown (2). The observables of T_1 and T_2 change as in (3) according to Lemma 24. □

The common minimisation makes two tree-PTS's more bisimilar — by strictly reducing the number of non-bisimilar states — but it also changes the observables. We can compensate for this by inserting two bisimilar sub-trees which compute the missing observables.

Proposition 26 *Given two tree-subPTS* T_1 *and* T_2 *such that* $T_1 \sim_{io} T_2$, *assume that the n-cutoffs are bisimilar, i.e.* $T_1|_n \sim_{tb} T_2|_n$, *and consider the common minimisation of* $T_1|_{n+1} \oplus T_2|_{n+1}$ *with respect to two non-bisimilar states* $s_1 \in T_1$ *and* $s_2 \in T_2$ *in layer* $n+1$, *then the insertion of the padded versions of the depleted and original version of* $s_{1\triangle}$ *and* $s_{2\triangle}$ *gives:*

$$\begin{aligned}T_1 &\sim_{io} (T_1^{\min} \lhd^{s_1}_{p_1-\min(p_1,p_2)} \textsc{Padding}(s_{1\triangle}, \widehat{s_{2\triangle}}))\\ T_1^{\min} &\sim_{tb} (T_1^{\min} \lhd^{s_1}_{p_1-\min(p_1,p_2)} \textsc{Padding}(s_{1\triangle}, \widehat{s_{2\triangle}}))\end{aligned}$$

$$\begin{aligned}T_2 &\sim_{io} (T_2^{\min} \lhd^{s_2}_{p_2-\min(p_1,p_2)} \textsc{Padding}(\widehat{s_{1\triangle}}, s_{2\triangle}))\\ T_2^{\min} &\sim_{tb} (T_2^{\min} \lhd^{s_2}_{p_2-\min(p_1,p_2)} \textsc{Padding}(\widehat{s_{1\triangle}}, s_{2\triangle}))\end{aligned}$$

PROOF. Obviously $T_i^{\min}$ and $(T_i^{\min} \lhd^{s_i}_{p_i-\min(p_1,p_2)} \textsc{Padding}(s_{1\triangle}, s_{2\triangle}))$, for $i = 1, 2$, are bisimilar by Lemma 20; and the same is rue if one of the sub-trees $s_{1\triangle}$ or $s_{2\triangle}$ is depleted. The observables of T_i where reduced by the common minimisation according to Lemma 24 and Proposition 25 by reducing the probabilities of some paths through $s_{i\triangle}$. Inserting the $s_{i\triangle}$ again with the missing probability compensates for this; while the depleted versions $\widehat{s_{i\triangle}}$ do not influence the observables. □

We can now show the main theorem establishing that the padding is correct.

Theorem 27 *Given two tree-PTS's T_1 and T_2 then* $\textsc{Padding}(T_1, T_2)$ *returns the tree-PTS's T_1' and T_2' such that $T_1' \sim_{tb} T_2'$, $T_1' \sim_{io} T_1$, and $T_2' \sim_{io} T_2$.*

PROOF. The proof is by induction on the height of $T_1 \oplus T_2$. The algorithm Padding progresses through the trees T_1 and T_2 layer by layer starting from the terminal states or leaves (base case). The sub-routine MaxLumping tries to identify non-bisimilar states/classes in T_1 and T_2 respectively. If a mismatch is found on layer $n + 1$ then the n-cutoffs are bisimilar, i.e. $T_1|_n \sim_{tb} T_1|_n$ (induction hypothesis).

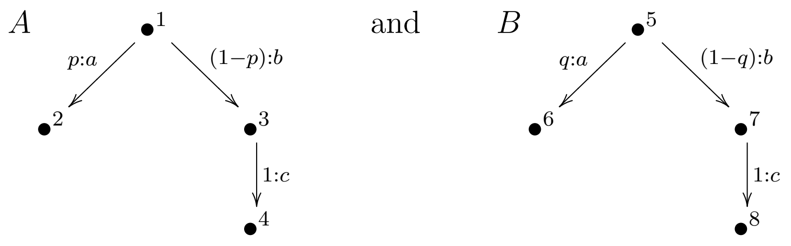
Calling the sub-routine MaxLumping has three possible outcomes: (i) no mismatches are found, (ii) there is only one mismatch state/class on level $n + 1$, i.e. one of the two trees is higher, or (iii) there are two states/classes s_1 and s_2 in T_1 and T_2 respectively which are not bisimilar. If case (i) holds, then T_1 and T_2 are already bisimilar and the algorithm terminates. In case (ii) we need to extend the shorter tree, i.e. assuming w.l.o.g. that T_1 is shorter, we need to construct a root extension $r \lhd_1^{t_0}\colon \omega T_1$. By Lemma 21 we have $T_1 \sim_{io} r \lhd_1^{t_0}\colon \omega T_1$, and as $T_1|_n$ and $T_2|_n$ are bisimilar we now also have that $T_1|_{n+1} \sim_{tb} T_2|_{n+1}$.

In the general case (iii) the transformation proceeds in two phases: In phase one we construct a common minimisation of T_1 and T_2 such the n-cutoff are still bisimilar, and such that the number of non-bisimilar states in the $n+1$-cutoff is strictly reduced, cf. Proposition 25. In phase two we insert subtrees which compensate for the missing observables (and also guarantee that the normalisation condition of proper PTS's are fulfilled); as the new subtrees are bisimilar we do not increase the number of non-bisimilar states, see Proposition 26. In other words: Phase one strictly reduced the number of non-bisimilar states while phase two does not increase it; and: Phase one changes the observables but phase two compensates again. Together we therefore have a monotone decrease of the number of non-bisimilar states while the observables remain unchanged.

As we have only finitely many states in a layer we will eventually reach the situation that there are no non-bisimilar states in layer $n+1$ or more generally in the $n+1$-cutoff. The algorithm therefore terminates in each layer and as we also have only finitely many layers the algorithm terminates with $T_1' \sim_{tb} T_2'$, $T_1' \sim_{io} T_1$, and $T_2' \sim_{io} T_2$. $\square$

6 A Simple Example

Consider two processes A and B (with $p \neq q$ and $p, q \notin \{0, 1\}$) defined as:



The aim is to construct two processes A' and B' such that $A' \sim_{tb} B'$ and $A \sim_{io} A'$ and $B \sim_{io} B'$. Note that A and B do not have the same I/O observables, i.e. $A \not\sim_{io} B$, and that we also do not require this for the transformed processes.

If we call MaxLumping(A, B) we have as first splitter the set of all states, i.e. $S_1 = \{1, 2, \ldots, 8\}$. The backward image of S_1, i.e. calling Splitting$(S_1, \{S_1\})$, allows us to distinguish the terminal nodes from 'internal' nodes, i.e. we get the partition $K_1 = \{\{2, 4, 6, 8\}, \{1, 3, 5, 7\}\}$. Obviously, both processes have representatives in each of the two blocks of this partition.

If we continue lumping using the splitter $S_2 = \{2, 4, 6, 8\}$, i.e. the terminal nodes, we get a refined partition $K_2 = \{\{2, 4, 6, 8\}, \{3, 7\}, \{1\}, \{5\}\}$. The probability of reaching S_2 from nodes 3 and 7 is the same (namely 1) while 1 and 5 reach the terminal nodes with different probabilities (namely p and q) and thus form two separate blocks.

The splitting K_2 now is not properly balanced, i.e. the class $\{1\}$ has no representatives in B and $\{5\}$ has no representatives in A. MaxLumping(A, B) will thus report $\{1\}$ and $\{5\}$ as critical classes.

In order to fix the 'imbalance' between A and B we have to do two things: (i) minimising/unifying the probabilities of transitions emitting from 1 and 5, and (ii) introducing subtrees X and Y to compensate for the missing outputs and to re-establish normalised probabilities.

- The first step thus is to transform A and B into the following form:

$$\begin{tikzcd} & \bullet^1 \arrow[dl, "\min(p,q):a"'] \arrow[dr, "\min(1-p,1-q):b"] & \\ \bullet^2 & & \bullet^3 \arrow[d, "1:c"] \\ & & \bullet^4 \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} & \bullet^5 \arrow[dl, "\min(p,q):a"'] \arrow[dr, "\min(1-p,1-q):b"] & \\ \bullet^6 & & \bullet^7 \arrow[d, "1:c"] \\ & & \bullet^8 \end{tikzcd}$$

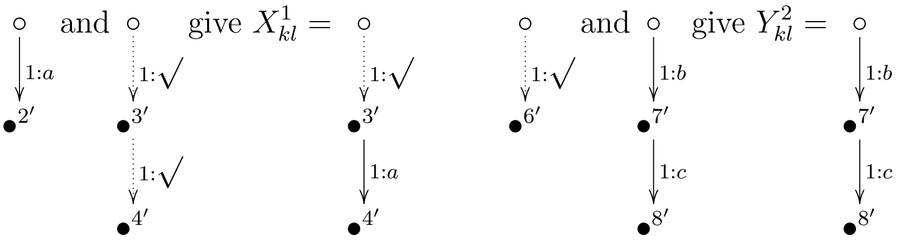
  These two transformed versions of A and B are now probabilistically tick bisimilar but unfortunately they have different outputs, i.e. they are not I/O equivalent; moreover they fail to fulfil the normalisation condition for PTS and Markov chains.
- To compensate for the "missing" probability $r = 1 - \min(p, q) - \min(1 - p, 1-q)$ we have to construct subtrees X and Y which reproduce the missing output states and which are bisimilar. As we have a critical class, namely $C_k = \{1\} \in A$ and $C_j = \{5\} \in B$, in each process we have to consider the third case (line 10) in PADDING. As the critical classes are singletons we have to take $s_1 = 1$ and $s_2 = 5$ in when we call FIXIT$(A, 1, B, 5)$.

  There are now for each of the states $s_1 = 1$ and $s_2 = 5$ two blocks they can make a move to, namely $B_k = \{3, 7\}$ and $B_l = \{2, 4, 6, 8\}$, but with different probabilities. We have now to chose in FIXIT (lines 10 to 13) representative states for these two blocks in each of the two processes A and B. In our example these states are uniquely determined, $t_{1k} = 3$, $t_{1l} = 2$, $t_{2k} = 7$, and $t_{2l} = 6$. Furthermore, we have to consider the following sub-trees of A and B which are rooted in these representative states:

$$T_{1l} = \begin{tikzcd} \circ \arrow[d, "1:a"] \\ \bullet^2 \end{tikzcd} \quad T_{1k} = \begin{tikzcd} \circ \arrow[d, "1:b"] \\ \bullet^3 \arrow[d, "1:c"] \\ \bullet^4 \end{tikzcd} \quad T_{2l} = \begin{tikzcd} \circ \arrow[d, "1:a"] \\ \bullet^6 \end{tikzcd} \quad T_{2k} = \begin{tikzcd} \circ \arrow[d, "1:b"] \\ \bullet^7 \arrow[d, "1:c"] \\ \bullet^8 \end{tikzcd}$$

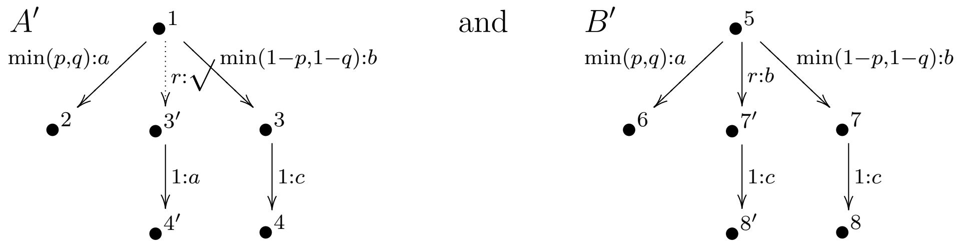
  We assume w.l.o.g. $p > q$, i.e. $\min(p, q) = q$ and $\min(1 - p, 1 - q) = 1 - p$ (the other case is symmetric) and thus $r_{1k} = r_{2l} = p - q$. The compensation tree X^1_{kl} for A thus must fulfil $X^1_{kl} \sim_{io} T_{1l}$ and for Y^2_{kl} in B we have $Y^2_{kl} \sim_{io} T_{2k}$ and furthermore $X^1_{kl} \sim_{tb} Y^2_{kl}$. In order to construct X^1_{kl} and Y^2_{kl} we thus just have to call (recursively) PADDING$(T_{1k}, \hat{T}_{1l})$ and PADDING$(\hat{T}_{2k}, T_{2l})$.

  We therefore have to compare twice two trees in order to obtain a padded version (calling PADDING recursively):



A dotted transition denotes a ticked or depleted transition: it means that no computation takes place during this transition (i.e. the state does not change) only time passes. Empty nodes ∘ denote copies of the dummy state: they can be seen as place-holders that inherit the state from the previous node. Obviously these two sub-trees X^1_{kl} and Y^2_{kl} fulfil the needed requirements regarding bisimilarity and I/O equivalence.

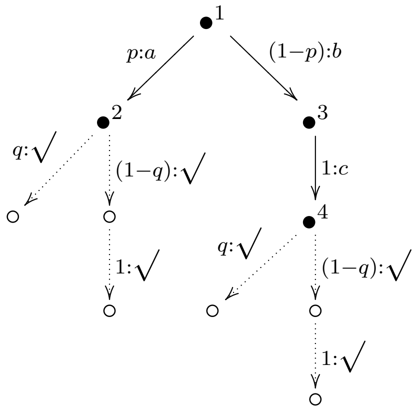
- Finally we have to introduce these compensatory sub-trees into the transformed versions of A and B to obtain (with $r = 1 - \min(p, q) - \min(1 - p, 1 - q)$):



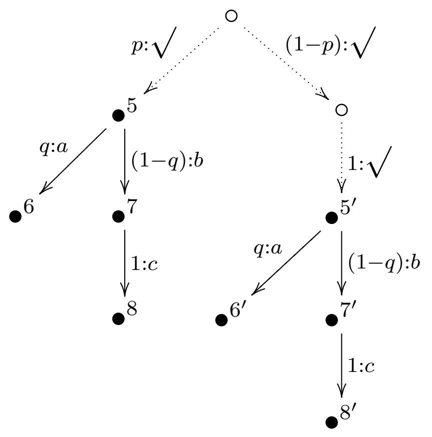
7 Comparisons and Related Work

7.1 Comparison with Agat's transformation

In [2] Agat proposes a transformation technique in the context of a type system with security types. The type system, besides detecting illegal information flow, transforms conditional statements that branch on high-security data into new statements where both conditional branches have been made bisimilar, thus making the new statement immune to timing attacks and the associated high-security data safe. The conditional statement is transformed by adding a 'dummy copy' of each branch to its counterpart such that, when the new branches are compared against each other, the original code is matched against its dummy copy. If we adapt this technique to PTS's and apply it to the processes A and B considered before, we get the transformed processes shown below:



and



According to Agat's transformation, a 'dummy execution' of B should follow A and a 'dummy execution' of A should precede B. Given that A has two final states, two dummy copies of B are appended to process A. Similarly, a dummy copy of A is put before B. However, in order to preserve the tree structure of the process, a new copy of B is made so it can be joined to one of the final states of the dummy copy of A.

Clearly the two transformed trees are bisimilar and the new processes are I/O-equivalent to their original versions. It is also clear that this transformation not only creates a greater number of states than our proposed solution but, more importantly, generates processes with longer execution time.

7.2 Related Literature

The work by Agat [2] is certainly the most closely related to ours; in fact we are not aware of other approaches to closing timing leaks which exploit program transformation techniques. Apart from the basic differences between our transformation algorithm and the Agat transformation type system explained in the previous section, our approach is in some sense more abstract than [2] in that it does not address a specific language; rather it is in principle adaptable to any probabilistic language whose semantics can be expressed in terms of our PTS model.

Timing attacks are prominently studied in security protocol analysis, and in general in the analysis of distributed programs specifically designed to achieve secure communication over inherently insecure shared media such as the Internet. In this setting, timing channels represent a serious threat as shown for example in [18] where a timing attack is mounted on RSA encryption, and in

[19] where it is shown how by measuring the time required by certain operation a web site can learn information about the user's past activities; or in [20] where an attack is described over Abadi's private authentication protocol [21].

The various approaches which have been proposed in the literature for the time analysis of security and cryptographic protocols mainly exploit languages and tools based on formal methods (timed automata, model checkers and process algebra). For example, [22] develop formal models and a timed automata based analysis for the Felten and Schneider's web privacy attack mentioned above. A process algebraic approach is adopted in [13] and [23] where the problem of timing information leakage is modelled as non-interference property capturing some time-dependent information flows. A different approach is the one adopted in [24] where static analysis techniques are used to verify communication protocols against timing attacks.

8 Conclusion and Further Work

In this paper we have investigated possible countermeasures against timing attacks in a process algebraic setting. The aim is to transform systems such that their timing behaviour becomes indistinguishable – i.e. they become bisimilar – while preserving their computational content – i.e. I/O behaviour. Our particular focus in this work was on finite, tree-like, probabilistic systems and probabilistic bisimulation.

Simple obfuscation methods have been consider before, e.g. by Agat [2], which may lead to a substantial increase of the (average) running time of the transformed processes. Our approach attempts to minimise this additional running time overhead introduced by the transformation. The padding algorithm achieves this aim by patching time leaks 'on demand' whenever the lumping algorithm which aims to establish the bisimilarity of two processes encounters a critical situation. Further work will be directed towards improving the padding algorithm further. At various stages of the algorithm we make nondeterministic choices (of certain states or classes); it will be important to investigate strategies to achieve optimal choices.

We also plan to investigate probabilistic and conditional padding. The current algorithm introduces time leak fixes whenever they appear to be necessary. The resulting processes are therefore perfectly bisimilar. However, one could also think of applying patches randomly or only if the expected time leak exceeds a certain threshold. One would then obtain approximate or ε-bisimilar systems – in the sense of our previous work [7,1] – but could expect a smaller increase of the (average) running time and/or the size of the system or program considered. Optimising the trade-off between vulnerability against timing attacks – measured quantitatively by ε – and the additional costs – running time, system size etc. – requires a better understanding of the relation between these different factors.

References

[1] A. Di Pierro, C. Hankin, H. Wiklicky, Measuring the confinement of probabilistic systems, Theoretical Computer Science 340 (1) (2005) 3–56.

[2] J. Agat, Transforming out timing leaks, in: POPL '00: Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM Press, New York, NY, USA, 2000, pp. 40–53.

[3] R. van Glabbeek, S. Smolka, B. Steffen, Reactive, generative and stratified models of probabilistic processes, Information and Computation 121 (1995) 59–80.

[4] R. Segala, N. Lynch, Probabilistic simulations for probabilistic processes, in: Proceedings of CONCUR 94, Vol. 836 of Lecture Notes in Computer Science, Springer Verlag, 1994, pp. 481–496.

[5] J. G. Kemeny, J. L. Snell, Finite Markov Chains, D. Van Nostrand Company, 1960.

[6] B. Jonsson, W. Yi, K. Larsen, Probabilistic Extentions of Process Algebras, Elsevier Science, Amsterdam, 2001, Ch. 11, pp. 685–710, see [25].

[7] A. Di Pierro, C. Hankin, H. Wiklicky, Quantitative relations and approximate process equivalences, in: D. Lugiez (Ed.), Proceedings of CONCUR'03, Vol. 2761 of Lecture Notes in Computer Science, Springer Verlag, 2003, pp. 508–522.

[8] B. Lampson, A Note on the Confinement Problem, Communications of the ACM 16 (10) (1973) 613–615.

[9] J. Goguen, J. Meseguer, Security Policies and Security Models, in: IEEE Symposium on Security and Privacy, IEEE Computer Society Press, 1982, pp. 11–20.

[10] J. Gray, III, Towards a mathematical foundation for information flow security, in: Proceedings of the 1991 Symposium on Research in Security and Privacy, IEEE, Oakland, California, 1991, pp. 21–34.

[11] P. Ryan, S. Schneider, Process algebra and non-interference, Journal of Computer Security 9 (1/2) (2001) 75–103, special Issue on CSFW-12.

[12] K. Larsen, A. Skou, Bisimulation through probabilistic testing, Information and Computation 94 (1991) 1–28.

[13] R. Focardi, R. Gorrieri, F. Martinelli, Real-time information flow analysis, IEEE Journal on Selected Areas in Communications 21 (1) (2003) 20–35.

[14] S. Derisavi, H. Hermanns, W. H. Sanders, Optimal state-space lumping in Markov chains, Information Processing Letters 87 (6) (2003) 309–315.

[15] R. Focardi, R. Gorrieri, Classification of security properties (Part I: Information Flow), in: Foundations of Security Analysis and Design, Tutorial Lectures, Vol. 2171 of Lecture Notes in Computer Science, Springer Verlag, 2001, pp. 331–396.

[16] R. Paige, R. Tarjan, Three partition refinement algorithms, SIAM Journal of Computation 16 (6) (1987) 973–989.

[17] S. Derisavi, H. Hermanns, W. H. Sanders, Optimal state-space lumping in Markov chains, Information Processing Letters 87 (6) (2003) 309–315.

[18] P. Kocher, Cryptanalysis of Diffie-Hellman, RSA, DSS, and other cryptosystems using timing attacks, in: D. Coppersmith (Ed.), Advances in Cryptology, CRYPTO '95, Vol. 963 of Lecture Notes in Computer Science, Springer-Verlag, 1995, pp. 171–183.

[19] E. W. Felten, M. A. Schneider, Timing attacks on web privacy, in: Proceedings of CCS'00, ACM, Athens, Greece, 2000, pp. 25–32.

[20] R. J. Corin, S. Etalle, P. H. Hartel, A. Mader, Timed analysis of security protocols, Journal of Computer Security.

[21] M. Abadi, Private authentication, in: R. Dingledine, P. F. Syverson (Eds.), 2nd Int. Workshop on Privacy Enhancing Technologies, Vol. 2482 of Lecture Notes in Computer Science, Springer Verlag, 2002, pp. 27–40.

[22] R. Gorrieri, R. Lanotte, A. Maggiolo-Schettini, F. Martinelli, S. Tini, E. Tronci, Automated analysis of timed security: A case study on web privacy, Journal of Information Security 2 (2004) 168–186.

[23] R. Gorrieri, F. Martinelli, A simple framework for real-time cryptographic protocol analysis with composition proof rules, Science of Computer Programming 50 (2004) 23–49.

[24] M. Buchholtz, S. Gilmore, J. Hillston, F. Nielson, Securing statically-verified communications protocols against timing attacks, in: 1st International Workshop on Practical Applications of Stochastic Modelling (PASM 2004), Vol. 128, 2005, pp. 123–143.

[25] J. Bergstra, A. Ponse, S. Smolka (Eds.), Handbook of Process Algebra, Elsevier Science, Amsterdam, 2001.