Approximated Computationally Bounded Simulation Relations for Probabilistic Automata[*]

Roberto Segala and Andrea Turrini
Dipartimento di Informatica
Università di Verona - Italy

Abstract

We study simulation relations for Probabilistic Automata that require transitions to be matched up to negligible sets provided that computation lengths are polynomially bounded. These relations are meant to provide rigorous grounds to parts of correctness proofs for cryptographic protocols that are usually carried out by semi-formal arguments. We illustrate our ideas by recasting a correctness proof of Bellare and Rogaway based on the notion of matching conversation.

1 Introduction

The *simulation method* [15] for hierarchical verification of concurrent systems consists of establishing relations between the states of two automata, called *simulation relations*, and to verify that such relations satisfy appropriate *step conditions*. Typically, a step condition requires that each transition of the simulated system can be matched by the simulating system up to the given simulation relation. The main advantage of the simulation method is the ability of reducing the analysis of global properties of computations to the analysis of local properties of transitions. Also, concepts like transitivity and compositionality studied in the context of process algebras [17] allow us to further decompose large problems into smaller problems and to verify systems hierarchically, that is, by building several intermediate refinements between specifications and implementations. Often hierarchical verification is simpler and cleaner than direct one-step verification. The simulation method has been extended to systems that include probability as well, leading to the model of probabilistic automata and to related notions of probabilistic simulations [24]. The reader interested in surveys of the extensive literature on related models and extensions is referred to [23, 26].

[*]Supported by INRIA project ProNoBiS and MIUR project AIDA.

In this paper we are interested in studying how the simulation method can be applied to the verification of protocols that involve cryptographic elements, seeking techniques that lead to simple, local and rigorous arguments of correctness. Our main motivation is the fact that in the crypto community several papers contain rigorous definitions of correctness that are proved via complex arguments about global properties of computations; often these arguments appear rather informal. In several cases the arguments involve showing correspondence between the computations of a real protocol and the computations of a more abstract, sometimes idealized, protocol that describes the expected behavior of a system. On the other hand, the simulation method is a rigorous sound technique to prove correspondence between computations of concrete and abstract systems. As a guideline we consider a message authentication protocol studied by Bellare and Rogaway with respect to the definition of correctness based on matching conversations [3].

Since cryptographic protocols involve probabilistic choices, we use probabilistic automata as basic underlying model. We describe a system in the Dolev-Yao style [9], where several agents interact via an adversarial network that records the past history and uses its knowledge to create, modify and/or forward messages between agents. At the abstract level we impose restrictions to the choices of the adversarial network so that correctness is guaranteed by definition, while at the concrete level we assume that the choices of the adversarial network are governed by a probabilistic polynomial time function. We then show that there is an appropriate simulation relation from the concrete to the abstract system. Our main goal here is to understand what such a simulation relation should look like and to provide evidence via a simple example of the potentials of using the simulation method.

The notion of simulation that we propose, called *polynomially accurate probabilistic simulation*, requires that a step of the concrete system is not necessarily matched exactly by the abstract system, but rather up to some error ε, which should correspond to the probability with which the adversarial network may be able to compromise correctness. The error should then be exponentially small in some security parameter provided that computations are of polynomial length. For this reason, our simulations are defined over families of probabilistic automata, parameterized over the security parameter, and relate computations rather than states to account for lengths of computations. Using the notion of lifting of a relation to probability measures (see e.g. [23]), we impose a step condition stating informally that whenever two probability measures over executions are related up to some error ε, where computations have polynomial length, each computational step from the first measure can be matched by a computational step from the second measure up to an extra error which is smaller than any polynomial. This means that the cumulated error in matching any polynomial number of steps is also smaller than any polynomial. Thus, the probability of the computations that cannot be matched, that is those that may lead to failure, is negligible.

The advantage of our notion of polynomially accurate probabilistic simulation, which we illustrate via the example of the MAP1 protocol of Bellare and Rogaway [3], is that the verification of the step condition reduces directly to the statement of correctness of the underlying cryptographic primitives. Also, the transitivity of our simulation relations allow us to separate concerns by describing adversarial networks at several levels of abstractions. For example, in the Bellare-Rogaway case study we can define a first intermediate network where all new nonces from agents are forced to be fresh, and a second abstract network where all new nonces from agents are fresh and at the same time no signature (message authentication code) is forged. The step condition for the first level can be proved by observing that nonces are generated randomly by agents, while the step condition for the second level follows from the observation that a transition that cannot be matched is a forger for the employed signature schema. Finally, the compositionality of our simulation relations allow us to analyze only subparts of a system. For example, the second level of the Bellare-Rogaway analysis can be carried out by ignoring the structure of the agents.

The simulation method is used already in the security literature. For example in [2] bisimulation relations are used to prove correctness of implementations according to the notion of reactive simulatability [20, 21]. Although the definition of bisimulation is not worked out in full details, the idea is clear: transitions should be matched up to some "error sets", where an error set is a set of parts of transitions (e.g., messages, states) that have no corresponding piece in the abstract system; then, a separate arguments shows that the global probability of the error sets is negligible. In our approach we impose conditions on the probabilities of the error sets directly in the step condition with the aim of showing that computations with non-negligible error sets are attackers for some underlying protocol or cryptographic primitive. It is worth investigating whether the proofs in [2] would benefit from the use of polynomially accurate simulation (or bisimulation) relations. Simulation relations are used also in [8] in the context of the Universally Composable framework [6]. In this case simulation relations are exact and the computational arguments are carried out with respect to a notion of approximated probabilistic language inclusion [7] based on the trace distribution semantics of [22]. Also in [18] there is a use of exact probabilistic bisimulations in the context of a probabilistic polynomial time process calculus. In this case the computational aspects are captured directly in the definition of the calculus. Another proposal of approximated probabilistic simulation relations appears in [19]. In this case a distance between probability of measures is defined based on the ability to produce similar trace distributions. Then, roughly speaking, an ε simulation matches steps from ε-distant measures by preserving ε-distance. Our definition is based on a different notion of distance and permits distances to grow by a negligible value at each step.

The idea of hierarchical analysis is also not new in the security literature. Besides the above literature on the use of the simulation method, in [25] there is an idea of representing a correctness proof as a sequence of related games, where games are representations of attacks against protocols that are either described at different levels of abstraction, and where two games are related if the difference of the probabilities of successful attacks is negligible. A similar idea is followed by [4, 5] with the difference that a mechanical correctness proof can be provided by means of a collection of sound game transformation rules. We view probabilistically accurate simulations as a potential tool for proving correctness of game transformations in [25] and for proving the soundness of the rules of [4, 5]. Indeed, the two steps of our case study are very similar in style to the game transformation of [25].

Another formal analysis of the MAP1 protocol of Bellare and Rogaway appears in [13]. In this case a probabilistic polynomial time observational equivalence is defined on a probabilistic polynomial time calculus similar to the asynchronous π-calculus: two processes are equivalent if for each context, the probability that the context distinguishes the two processes is negligible. The calculus is used to formalize the MAP1 protocol, while the analysis is carried out via typical arguments from the cryptography literature. More recent work [18] considers also axiomatizations for the observational equivalence, which could be used as an alternative process algebraic method to prove the correctness of the MAP1 protocol.

Though our main goal was to give more rigorous grounds to proofs carried out in the pure computational framework, we believe our results can be seen as a further step in the process of proving the soundness of the Dolev-Yao model [9] with respect to the computational model. Indeed, in our case study the abstract description of the adversarial network assumes perfect cryptography and fresh nonces. Initial work on the soundness of the Dolev-Yao model [1] was considering passive eavesdropping, showing how non-derivability of terms in the Dolev-Yao model is related to non-computability in probabilistic polynomial time in the concrete model. Later work started to consider active adversaries as well [2, 12, 16]. In this last case it is important to establish connections between concrete and abstract computations, which is one of the uses of polynomially accurate simulation relations. One point of our definition is that we can work with any abstract model.

The rest of the paper is structured as follows. Section 2 gives formal definitions of probabilistic automata; Section 3 introduces the polynomially accurate probabilistic simulation relation and its properties; Section 4 shows the application of polynomially accurate probabilistic simulation to the Bellare-Rogaway protocol; Section 5 gives some concluding remarks.

2 Probabilistic Automata

In this section we recall the basic definitions for Probabilistic Automata and the notion of simulation of [24]. The reader interested in an introduction to Probabilistic Automata is referred to [23].

2.1 Mathematical Preliminaries

A *σ-field* over a set X is a set $\mathcal{F} \subseteq 2^X$ that includes X and is closed under complement and countable union. A *measurable space* is a pair $(X, \mathcal{F})$ where X is a set, also called *sample space*, and $\mathcal{F}$ is a σ-field over X. A measurable space $(X, \mathcal{F})$ is called *discrete* if $\mathcal{F} = 2^X$. A *measure* over a measurable space $(X, \mathcal{F})$ is a function $\rho \colon \mathcal{F} \to \mathbb{R}^{\geqslant 0}$ such that, for each countable collection $\{X_i\}_{i \in I}$ of pairwise disjoint elements of $\mathcal{F}$, $\rho(\cup_I X_i) = \sum_I \rho(X_i)$. A *probability measure* over a measurable space $(X, \mathcal{F})$ is a measure ρ over $(X, \mathcal{F})$ such that $\rho(X) = 1$. A *sub-probability measure* over $(X, \mathcal{F})$ is a measure over $(X, \mathcal{F})$ such that $\rho(X) \leqslant 1$. A measure over a discrete measurable space $(X, 2^X)$ is called a *discrete measure* over X. The *support* of a measure ρ over $(X, \mathcal{F})$ is the set $\{x \in X \mid \rho(x) > 0\}$.

Given a set X, denote by *Disc*(X) the set of discrete probability measures over X, and by *SubDisc*(X) the set of discrete sub-probability measures over X. We call a discrete probability measure a *Dirac* measure if it assigns measure 1 to exactly one object x (denote this measure by δ_x). We also call Dirac a sub-probability measure that assigns measure 0 to all objects. In the sequel discrete sub-probability measures are used to describe progress. If the measure of a sample space is not 1, then it means that with some non-zero probability the system does not progress.

2.2 Probabilistic Automata

A *Probabilistic Automaton* (PA) is a tuple $(S, \bar{s}, A, D)$ where S is a countable set of *states*, $\bar{s} \in S$ is the *start state*, A is a set of *actions*, and $D \subseteq S \times A \times \textit{Disc}(S)$ is a *transition relation*.

The restriction on the state set S to be countable is not necessary for the results if this paper, but we impose it for simplicity.

Throughout the paper we let $\mathcal{A}$ range over probabilistic automata, q, r, s range over states, a, b, c range over actions, and μ range over discrete measures over states. We also denote the generic elements of a probabilistic automaton $\mathcal{A}$ by S, $\bar{s}$, A, D, and we propagate primes and indices when necessary. Thus, for example, the probabilistic automaton $\mathcal{A}'_i$ has transition relation D'_i. We also denote the start state of a probabilistic automaton $\mathcal{A}, \mathcal{B}, \ldots$ by $\bar{a}, \bar{b}, \ldots$, respectively.

An element of a transition relation D is called a *transition* or a *step*. A transition *tr* $= (s, a, \mu)$, denoted alternatively by $s \xrightarrow{a} \mu$, is said to *leave* from state s, denoted also by *src*(*tr*), to be *labeled* by a, denoted by *act*(*tr*), and to *lead* to μ, denoted by *trg*(*tr*) or μ_{tr}. We also say that state s *enables* action a, that action a is *enabled* from s, and that (s, a, μ) is enabled from s.

An *execution fragment* of a PA $\mathcal{A}$ is a sequence of alternating states and actions, $\alpha = s_0 a_1 s_1 \ldots$, starting with a state and, if the sequence is finite, ending with a state, such that, for each non final index i, there exists a transition $(s_i, a_{i+1}, \mu_{i+1})$ in D with $\mu_{i+1}(s_{i+1}) > 0$. We denote the first state s_0 of α by *fstate*(α). We say that an execution fragment is *finite* if it is a finite sequence, and we denote the last state of a finite execution fragment α by *lstate*(α). We define the length of an execution fragment α to be the number of occurrences of actions in α. An *execution* of a PA $\mathcal{A}$ is an execution fragment of $\mathcal{A}$ whose first state is $\bar{s}$. We denote by *Frags*$^*(\mathcal{A})$ the set of finite execution fragments of $\mathcal{A}$, by *Frags*$(\mathcal{A})$ the set of finite or infinite execution fragments, and by *Execs*$^*(\mathcal{A})$, *Execs*$(\mathcal{A})$ the corresponding sets of executions. We let ν range over discrete probability measures over finite executions of $\mathcal{A}$, that is, $\nu \in \textit{Disc}(\textit{Execs}^*(\mathcal{A}))$.

A scheduler for a PA $\mathcal{A}$ is a function $\sigma \colon \textit{Frags}^*(\mathcal{A}) \to \textit{SubDisc}(D)$ such that, for each finite execution fragment α and each transition *tr* with $\sigma(\alpha)(\textit{tr}) > 0$, $\textit{src}(\textit{tr}) = \textit{lstate}(\alpha)$. A scheduler σ is deterministic if, for each finite execution fragment α, $\sigma(\alpha)$ is a Dirac sub-measure.

A scheduler σ can be used to describe the result of resolving nondeterminism starting from some state s. Specifically, a scheduler σ and a state s induce a probability measure $\varepsilon_{\sigma,s}$ over execution fragments as follows. The basic measurable events are the cones of finite execution fragments, where the cone of a finite execution fragment α, denoted by C_α, is the set $\{\alpha' \in \mathit{Frags}(\mathcal{A}) \mid \alpha \leqslant \alpha'\}$, where $\leqslant$ is the standard prefix preorder on sequences. The probability $\varepsilon_{\sigma,s}$ of a cone C_α is defined recursively as follows:

$$\varepsilon_{\sigma,s}(C_\alpha) = \begin{cases} 0 & \text{if } \alpha = q \text{ for some state } q \neq s, \\ 1 & \text{if } \alpha = s, \\ \varepsilon_{\sigma,s}(C_{\alpha'}) \sum_{\mathit{tr} \in D(a)} \sigma(\alpha')(\mathit{tr})\mu_{\mathit{tr}}(q) \\ & \text{if } \alpha = \alpha' a q, \end{cases}$$

where $D(a)$ denotes the set of transitions of D with label a. Standard measure theoretical arguments ensure that $\varepsilon_{\sigma,s}$ extends uniquely to the σ-field generated by cones. We call the measure $\varepsilon_{\sigma,s}$ a *probabilistic execution fragment* of $\mathcal{A}$ and we say that it is generated by σ from s. If s is the start state of $\mathcal{A}$, then we say that $\varepsilon_{\sigma,s}$ is a *probabilistic execution*.

We now turn to the notion of simulation for probabilistic automata [24], defining first what it means to lift a relation on states to a relation on measures. Let $\mathcal{R}$ be a relation from a set X to a set Y. The lifting of $\mathcal{R}$, denoted by $\mathcal{L}(\mathcal{R})$, is a relation from $\mathit{Disc}(X)$ to $\mathit{Disc}(Y)$ such that $\rho_1\ \mathcal{L}(\mathcal{R})\ \rho_2$ if and only if there exists a *weighting function* $w\colon X \times Y \to [0,1]$ such that (1) $w(x_1,x_2) > 0$ implies $x_1\ \mathcal{R}\ x_2$, (2) $\sum_{x_1} w(x_1,x_2) = \rho_2(x_2)$, and (3) $\sum_{x_2} w(x_1,x_2) = \rho_1(x_1)$. An alternative definition of lifting given in a more probabilistic style states that $\rho_1\ \mathcal{L}(\mathcal{R})\ \rho_2$ iff there exists a joint measure w with marginal measures ρ_1 and ρ_2 such that the support of w is included in $\mathcal{R}$.

A *simulation* from a PA $\mathcal{A}_1$ to PA $\mathcal{A}_2$ is a relation $\mathcal{R}$ from S_1 to S_2 such that

- $\bar{s}_1\ \mathcal{R}\ \bar{s}_2$ and
- for each pair $(s_1, s_2) \in\mathcal{R}$, if $(s_1, a, \mu_1) \in D_1$, then there exists $(s_2, a, \mu_2) \in D_2$ such that $\mu_1\ \mathcal{L}(\mathcal{R})\ \mu_2$.

We say that $\mathcal{A}_1$ is simulated by $\mathcal{A}_2$, denoted by $\mathcal{A}_1 \preceq \mathcal{A}_2$, if there exists a simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$.

It is known that relation $\preceq$ is transitive and preserved by parallel composition of PAs. This is the key feature that enables hierarchical and modular verification. We do not define composition here and we refer the interested reader to [23].

3 Polynomially Accurate Simulations

In this section we define our notion of polynomially accurate simulation relation. Our aim is to define a relation where transitions are matched up to some error that is smaller than any polynomial in some security parameter k provided that computations are of polynomial length. This means that we need a relation that can "see" lengths of computations, a notion of lifting that accounts for errors, and a notion of security parameter. Furthermore, since we will also need ways to match sequences of steps, we need a way to bound the amount of extra error introduced by each step. We try to address one issue at a time, getting closer and closer to our desired notion of simulation.

The first step is to define a relation that can ”see” lengths of computation. For this purpose, we define a relation on sets of executions rather than sets of states. This definition is based on a derived notion of transition that shows how finite executions evolve in a single step. Formally, we say that there is a step from a finite execution α to a measure $\nu \in \mathit{Disc}(\mathit{Execs}^*(\mathcal{A}))$, denoted by $\alpha \longrightarrow \nu$, if there exists a transition $(\mathit{lstate}(\alpha), a, \mu)$ such that, for each finite execution $\alpha a s$, $\nu(\alpha a s) = \mu(s)$.

Definition 1. *An* execution simulation *from a PA $\mathcal{A}_1$ to a PA $\mathcal{A}_2$ is a relation $\mathcal{R}$ from $\mathit{Execs}^*(\mathcal{A}_1)$ to $\mathit{Execs}^*(\mathcal{A}_2)$ such that:*

- *$\bar{s}_1\ \mathcal{R}\ \bar{s}_2$ and*
- *for each pair $(\alpha_1, \alpha_2) \in\mathcal{R}$, if $\alpha_1 \longrightarrow \nu_1$, then there exists ν_2 such that $\alpha_2 \longrightarrow \nu_2$ and $\nu_1\ \mathcal{L}(\mathcal{R})\ \nu_2$.*

We say that $\mathcal{A}_1$ is execution simulated by $\mathcal{A}_2$, denoted by $\mathcal{A}_1 \preceq_e \mathcal{A}_2$, if there exists an execution simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$.

It is interesting to observe that so far we have not introduced anything new since $\preceq$ and $\preceq_e$ coincide.

Proposition 1. *Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two PAs. Then $\mathcal{A}_1 \preceq_e \mathcal{A}_2$ if and only if $\mathcal{A}_1 \preceq \mathcal{A}_2$.*

Proof outline. Given a simulation relation $\mathcal{R}$ from S_1 to S_2, define a relation $\mathcal{R}'$ from $\mathit{Execs}^*(\mathcal{A}_1)$ to $\mathit{Execs}^*(\mathcal{A}_2)$ such that $\alpha_1\ \mathcal{R}'\ \alpha_2$ iff $\mathit{lstate}(\alpha_1)\ \mathcal{R}\ \mathit{lstate}(\alpha_2)$. It is routine to verify that $\mathcal{R}'$ is an execution simulation.

Conversely, given an execution simulation $\mathcal{R}$ from $\mathit{Execs}^*(\mathcal{A}_1)$ to $\mathit{Execs}^*(\mathcal{A}_2)$, define a relation $\mathcal{R}'$ from S_1 to S_2 such that $s_1\ \mathcal{R}'\ s_2$ iff there exist α_1 and α_2 such that $\alpha_1\ \mathcal{R}\ \alpha_2$, $\mathit{lstate}(\alpha_1) = s_1$, and $\mathit{lstate}(\alpha_2) = s_2$. It is routine to verify that $\mathcal{R}'$ is a simulation relation. $\square$

We now generalize the notion of lifting so that two measures are not related exactly, but up to some error ε. Our definition states that two measures are related up to ε if some $(1-\varepsilon)$ fractions of the two measures are related exactly.

Definition 2. *Let $\mathcal{R}$ be a relation from X to Y and let $\varepsilon \geqslant 0$. The* ε-lifting *of $\mathcal{R}$, denoted by $\mathcal{L}(\mathcal{R}, \varepsilon)$ is a relation from $\mathit{Disc}(X)$ to $\mathit{Disc}(Y)$ defined as follows: for each pair ρ_x and ρ_y of probability measures on X and Y, respectively,*

- *if $\varepsilon > 1$, then $\rho_x\ \mathcal{L}(\mathcal{R}, \varepsilon)\ \rho_y$;*

- *if $\varepsilon \in [0,1]$, then $\rho_x\ \mathcal{L}(\mathcal{R},\varepsilon)\ \rho_y$ if there exist $\rho'_x, \rho''_x \in$ Disc(X) and $\rho'_y, \rho''_y \in$ Disc(Y) such that*
  - $\rho_x = (1-\varepsilon)\rho'_x + \varepsilon\rho''_x$,
  - $\rho_y = (1-\varepsilon)\rho'_y + \varepsilon\rho''_y$,
  - $\rho'_x\ \mathcal{L}(\mathcal{R})\ \rho'_y$.

It is interesting to observe that ε-lifting is monotone on ε and that 0-lifting coincides with lifting.

Proposition 2. *The following holds for a relation $\mathcal{R}$ from X to Y.*

1. *$\mathcal{L}(\mathcal{R},0)$=$\mathcal{L}(\mathcal{R})$.*
2. *If $\varepsilon \leqslant \varepsilon'$, then $\mathcal{L}(\mathcal{R},\varepsilon)$⊆$\mathcal{L}(\mathcal{R},\varepsilon')$.*

The introduction of errors in execution simulations is then straightforward.

Definition 3. *An* ε-simulation *from a PA $\mathcal{A}_1$ to a PA $\mathcal{A}_2$ is a relation $\mathcal{R}$ from Execs$^*(\mathcal{A}_1)$ to Execs$^*(\mathcal{A}_2)$ such that:*

- *$\bar{s}_1\ \mathcal{R}\ \bar{s}_2$ and*
- *for each pair $(\alpha_1,\alpha_2)\ \in\mathcal{R}$, if $\alpha_1 \longrightarrow \nu_1$, then there exists ν_2 such that $\alpha_2 \longrightarrow \nu_2$ and $\nu_1\ \mathcal{L}(\mathcal{R},\varepsilon)\ \nu_2$.*

The above definition is still not adequate for handling cryptographic protocols. The point is that we desire to reach a point where the parts of transitions that cannot be matched correspond to bad behavior like guessing a key or forging a signature. Given a finite execution α there is always a way to resolve nondeterminism so that a key is guessed; what is difficult to do is to guess a key once we have a probability measure over executions obtained by generating a key. This suggests that our step conditions should be based on measures over executions rather than single executions. Furthermore, it is convenient to consider also pairs of measures that are related up to some error γ and limit the increment of the error. This leads to a new proposal of simulation relation that we define below. However, we first need to extend the notation for transitions to measures over executions.

Given a PA $\mathcal{A}$ and two measures $\nu,\nu' \in$ *Disc*(*Execs*$^*(\mathcal{A})$), we say that there exists a transition from ν to ν', denoted by $\nu \longrightarrow \nu'$, if there exists a scheduler σ such that for each finite execution $\alpha a s$, $\nu'(C_{\alpha a s}) = \nu(C_{\alpha a s}) + \nu(C_\alpha)\sum_{tr\in D(a)}\sigma(\alpha)(tr)\cdot\mu_{tr}(s)$, where $D(a)$ denotes the set of transitions with action a.

Definition 4. *Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two PAs and let $\mathcal{R}$ be a relation from Execs$^*(\mathcal{A}_1)$ to Execs$^*(\mathcal{A}_2)$. We say that $\mathcal{R}$ is an ε-*execution simulation *from $\mathcal{A}_1$ to $\mathcal{A}_2$ if*

1. *$\bar{s}_1\ \mathcal{R}\ \bar{s}_2$;*
2. *for each $\gamma \geqslant 0$, $\nu_1 \in$ Disc(Execs$^*(\mathcal{A}_1)$) and $\nu_2 \in$ Disc(Execs$^*(\mathcal{A}_2)$), if*
   - *$\nu_1\ \mathcal{L}(\mathcal{R},\gamma)\ \nu_2$,*
   - *$\nu_1 \longrightarrow \nu'_1$*

   then there exists ν'_2 such that
   - *$\nu_2 \longrightarrow \nu'_2$,*
   - *$\nu'_1\ \mathcal{L}(\mathcal{R},\gamma+\varepsilon)\ \nu'_2$.*

We say that $\mathcal{A}_1$ is ε-execution simulated by $\mathcal{A}_2$, denoted by $\mathcal{A}_1 \preceq^{\varepsilon}_{e} \mathcal{A}_2$, if there exists an ε-execution simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$.

The definition above satisfies few important properties. The first property is transitivity up to some combination of the errors; the second property is that sequences of n steps can be matched up to an error $n\varepsilon$. Thus, the bound on the error for a single step induces bounds on the error for any number of steps.

Proposition 3. *Let $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ be three PAs such that $\mathcal{A}_1 \preceq^{\varepsilon_{12}}_{e} \mathcal{A}_2$ and $\mathcal{A}_2 \preceq^{\varepsilon_{23}}_{e} \mathcal{A}_3$ for some $\varepsilon_{12},\varepsilon_{23} \geqslant 0$. Then $\mathcal{A}_1 \preceq^{\varepsilon_{13}}_{e} \mathcal{A}_3$ where $\varepsilon_{13} = \varepsilon_{12}+\varepsilon_{23}$.*

Proof sketch. Let $\mathcal{R}_{12}$, $\mathcal{R}_{23}$ be two ε-execution simulations that justify $\mathcal{A}_1 \preceq^{\varepsilon_{12}}_{e} \mathcal{A}_2$ and $\mathcal{A}_2 \preceq^{\varepsilon_{23}}_{e} \mathcal{A}_3$, respectively. Define $\mathcal{R}_{13}\subseteq$ *Execs*$^*(\mathcal{A}_1) \times$ *Execs*$^*(\mathcal{A}_3)$ as $\mathcal{R}_{13}= \{(\alpha_1,\alpha_3) \mid \exists\alpha_2 \in$ *Execs*$^*(\mathcal{A}_2).\alpha_1\ \mathcal{R}_{12}\ \alpha_2 \wedge \alpha_2\ \mathcal{R}_{23}\ \alpha_3\}$.

The condition on the start states is immediate since by hypothesis $\bar{s}_1\ \mathcal{R}_{12}\ \bar{s}_2$ and $\bar{s}_2\ \mathcal{R}_{23}\ \bar{s}_3$, hence by definition of $\mathcal{R}_{13}$, $\bar{s}_1\ \mathcal{R}_{13}\ \bar{s}_3$, as required.

The step condition is more involved and requires several technical constructions that we cannot include here. Suppose that $\nu_1\ \mathcal{L}(\mathcal{R}_{13},\gamma)\ \nu_3$, and suppose $\nu_1 \longrightarrow \nu'_1$. We first apply the definition of ε-lifting to decompose ν_1 and ν_3 into the parts that match exactly and those that do not match. Similarly we decompose $\nu_1 \longrightarrow \nu'_1$ into two parts. We then find a measure ν_2 that can be decomposed as well and use the step condition with $\gamma = 0$ from the matching parts of ν_1,ν_2,ν_3. Finally, we recompose pieces together. □

For the next property we first have to define what it means to reach a measure within n steps.

Definition 5. *Let $\mathcal{A}$ be a probabilistic automaton and σ be a scheduler for $\mathcal{A}$.*

We say that ν is a probability measure reached in at most n steps via σ if there is a sequence of probability measures $\nu_0,\ldots,\nu_n$ such that $\nu_0(\bar{s}) = 1$, $\nu_n = \nu$ and for each $0 \leqslant i < n$, σ schedules the transition $\nu_i \longrightarrow \nu_{i+1}$.

Proposition 4. *Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two PAs such that $\mathcal{A}_1 \preceq^{\varepsilon}_{e} \mathcal{A}_2$ for some $\varepsilon \geqslant 0$. Let $\mathcal{R}$ be an ε-execution simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$.*

For each scheduler σ_1 for $\mathcal{A}_1$, if ν_1 is reached via σ_1 within n steps, then there exists a scheduler σ_2 for $\mathcal{A}_2$ that reaches, within n steps, a probability measure ν_2 such that $\nu_1\ \mathcal{L}(\mathcal{R},n\varepsilon)\ \nu_2$.

Proof sketch. The proof is a classical inductive argument. The base case is trivial since start states are related and the Dirac measures over them are the only measures reachable within 0 steps. Hence such measures are related by $\mathcal{L}(\mathcal{R}, 0)$, as required. For the inductive step, by hypothesis we start from two measures ν_1 and ν_2 such that $\nu_1\ \mathcal{L}(\mathcal{R}, n\varepsilon)\ \nu_2$ where n is the number of steps used to reach them. By the step condition we have that after another step the reached measures ν'_1 and ν'_2 satisfy $\nu'_1\ \mathcal{L}(\mathcal{R}, n\varepsilon + \varepsilon)\ \nu'_2$, as required. □

We are now left with the computational aspects of our definition. For the purpose we talk about families of PAs and families of relations parameterized over a security parameter k. Furthermore, we impose the step condition only for measures that are reachable within a number of steps that is polynomial in k.

Definition 6. *Let $\{\mathcal{A}_k\}_{k\in K}$ and $\{\mathcal{B}_k\}_{k\in K}$ be two families of probabilistic automata; let $\mathcal{R}= \{\mathcal{R}_k\}_{k\in K}$ be a family of relations such that, for each $k \in K$, $\mathcal{R}_k$ is a relation from Execs$^*(\mathcal{A}_k)$ to Execs$^*(\mathcal{B}_k)$; let Poly be the set of positive polynomials over $\mathbb{N}$.*

We say that $\mathcal{R}$ is a polynomially accurate simulation *from $\{\mathcal{A}_k\}_{k\in K}$ to $\{\mathcal{B}_k\}_{k\in K}$ if*

1. *for each k, it holds that $\bar{a}_k\ \mathcal{R}_k\ \bar{b}_k$;*
2. *for each $c \in \mathbb{N}$ and for each $p \in$ Poly, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$, for all probability measures ν_1 and ν_2 and for each $\gamma \geqslant 0$, if*
   - *ν_1 is reached in at most $p(k)$ steps in $\mathcal{A}_k$,*
   - *$\nu_1\ \mathcal{L}(\mathcal{R}_k, \gamma)\ \nu_2$,*
   - *$\nu_1 \longrightarrow \nu'_1$*

   then there exists ν'_2 such that
   - *$\nu_2 \longrightarrow \nu'_2$,*
   - *$\nu'_1\ \mathcal{L}(\mathcal{R}_k, \gamma + k^{-c})\ \nu'_2$.*

We write $\{\mathcal{A}_k\}_{k\in K} \lesssim \{\mathcal{B}_k\}_{k\in K}$ if there exists a polynomially accurate simulation $\mathcal{R}$ from $\{\mathcal{A}_k\}_{k\in K}$ to $\{\mathcal{B}_k\}_{k\in K}$.

Finally we can use Proposition 4 to derive our main result, that is, existence of a polynomially accurate simulation allows us to match any polynomial number of steps with an error that is bounded by any polynomial.

Theorem 1. *Let $\{\mathcal{A}_k\}_{k\in K}$ and $\{\mathcal{B}_k\}_{k\in K}$ be two families of PAs such that $\{\mathcal{A}_k\}_{k\in K} \lesssim \{\mathcal{B}_k\}_{k\in K}$. Let $\mathcal{R}= \{\mathcal{R}_k\}_{k\in K}$ be a polynomially accurate simulation from $\{\mathcal{A}_k\}_{k\in K}$ to $\{\mathcal{B}_k\}_{k\in K}$.*

For each $c \in \mathbb{N}$, $p \in$ Poly, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$ and each scheduler σ_a for $\mathcal{A}_k$, if ν_a is the probability measure induced by σ_a after $p(k)$ steps, then there exists a scheduler σ_b for $\mathcal{B}_k$ that reaches, after $p(k)$ steps, a probability measure ν_b such that $\nu_a\ \mathcal{L}(\mathcal{R}_k, p(k)k^{-c})\ \nu_b$.

Proof sketch. The proof follows the lines of Proposition 4 with $\varepsilon = k^{-c}$ and $\bar{k}$ chosen according to the statement of Definition 6. □

4 A Simple Case Study

We illustrate the use of polynomially accurate simulations via a simple case study that deals with the Mutual Authentication Protocol MAP1 of Bellare and Rogaway [3] (cf. Fig 1). The protocol uses nonces to guarantee freshness and pseudorandom functions as message authentication tool. We first give some preliminary high level definitions of nonces, pseudorandom functions, message authentication codes, and forgers. These definitions are taken or adapted from [10,11]. Then we describe the MAP1 protocol and the structure of our correctness proof. Finally, we illustrate some of the details of the correctness proof, where we emphasize how the negation of the step condition of a polynomially accurate simulation corresponds to the definition of an attacker for the underlying cryptographic primitive or protocol.

4.1 Cryptographic Components

In the following we assume that k is a security parameter and that *Poly* is the set of positive polynomials over $\mathbb{N}$.

4.1.1 Nonces

A *nonce* of length k is an element of $\{0,1\}^k$ that is used at most once. An ideal way to satisfy unicity of nonces is to use a repository that keeps track of the nonces distributed in the past and that responds to all requests by returning a new value each time. The practical way to satisfy the unicity of nonces is to choose them randomly from $\{0,1\}^k$. In this way, if we choose randomly two nonces of length k, the probability that they are the same is at most 2^{-k}. This means that:

Claim 1. *For each $c \in \mathbb{N}$ and $p \in$ Poly, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$, if we choose randomly $n_1, \dots, n_{p(k)}$ nonces from $\{0,1\}^k$, then* ***Pr***$[n_i = n_j \mid i \neq j] < k^{-c}$.

4.1.2 Pseudorandom Functions

A *pseudorandom function* P is a function that can not be distinguished from a truly random function R by any efficient procedure (e.g., probabilistic polynomial time algorithm) that can get the values of both P and R at arguments of its choice. In other words, given a pseudorandom function P and a truly random function R, if we evaluate them on a polynomial number of values, then we are not able to distinguish when we are interacting with P or with R better than flipping a coin to decide.

Formally, we say that $\{f_s\colon \{0,1\}^* \to \{0,1\}^k\}_{s\in\{0,1\}^*}$ is a *pseudorandom function* if the following two conditions hold:

1. There exists a polynomial time algorithm that on inputs s and $x \in \{0,1\}^*$ returns $f_s(x)$.

2. For every probabilistic polynomial time machine M that samples values from a function f and returns a value in $\{0,1\}$, every $p \in$ *Poly* and all sufficiently large n's,

$$|\boldsymbol{Pr}[M^{F_n}(1^n) = 1] - \boldsymbol{Pr}[M^{H_n}(1^n) = 1]| < \frac{1}{p(n)}$$

   where F_n is a random variable uniformly distributed over the multi-set $\{f_s\}_{s\in\{0,1\}^n}$, H_n is a random variable uniformly distributed among all functions mapping arbitrarily long strings to k-long strings, $\boldsymbol{Pr}[M^{F_n}(1^n) = 1]$ is the probability that the machine M, on input 1^n, answers 1 provided that f is chosen according to F_n, and $\boldsymbol{Pr}[M^{H_n}(1^n) = 1]$ is the probability that the machine M, on input 1^n, answers 1 provided that f is chosen according to H_n.

This definition of pseudorandom function conceals technical aspects that are out the scope of this paper. Interested readers can find a justification of such technicalities and a generalized definition of pseudorandom functions in Section 3.6 of [10].

4.1.3 Message Authentication Code

A *message authentication scheme* is a triple (G, A, V) of probabilistic polynomial time algorithms satisfying the following two conditions:

1. On input 1^k, algorithm G (called the key-generator) outputs a bit string.

2. For every s in the range of $G(1^k)$ and for every $\alpha \in \{0,1\}^*$, algorithms A (authentication) and V (verification) satisfy $\boldsymbol{Pr}[V(s,\alpha,A(s,\alpha)) = 1] = 1$ where the probability is taken over the internal coin tosses of algorithms A and V.

We call $A(s,\alpha)$ a *message authentication code* (MAC) to the document α produced using the key s.

A *forger* is a process that, on input 1^k, can obtain message authentication codes to strings of its choice, relative to a key s that is generated by $G(1^k)$ and that the forger does not know. Such a forger is said to *succeed* (in existential forgery) if it outputs a valid MAC to a string for which it has not requested an authentication during the attack. That is, the forger is successful if it outputs a pair (α,β) such that $V(s,\alpha,\beta) = 1$ and α is different from all strings for which an authentication has been required during the attack. A message authentication scheme is *secure* (or unforgeable) if every feasible forger succeeds with at most negligible probability.

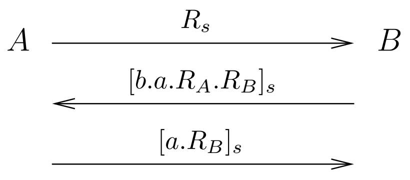

Figure 1. MAP1 protocol.

A way to construct message authentication schemes is to use pseudorandom functions, using the following construction (cf. Construction 6.3.1 of [11]): let $\{f_s\}_{s\in\{0,1\}^*}$ be a pseudorandom function. We define a message authentication scheme (G, A, V) as follows:

- Key generation with G: on input 1^k, we uniformly select $s \in \{0,1\}^k$ and output the key s.

- Authentication with A: on input a key $s \in \{0,1\}^k$ and a string $\alpha \in \{0,1\}^*$, we compute and output $f_s(\alpha)$ as an authentication of α.

- Verification with V: on input a key $s \in \{0,1\}^k$, a string $\alpha \in \{0,1\}^*$, and an alleged authentication β, we accept if and only if $\beta = f_s(\alpha)$.

Given a key s, we say that $f_s(m)$ is the message authentication code of m with respect to the key s and that f_s is a MAC value generator.

Proposition 5 (cf. Proposition 6.3.2 of [11]). *Suppose that $\{f_s\}_{s\in\{0,1\}^*}$ is a pseudorandom function. Then the given construction constitutes a secure message authentication scheme.*

A message authentication code can be used when an entity A wants to prove its identity to another entity B. If A and B share a secret key s and a pseudorandom function, then A can provide evidence of its identity by sending a message of the form $(a.m, f_s(a.m))$ to B, where m is some random value, a is a coding of the identity of A, and $a.m$ is the concatenation of a and m. B can rely on A's identity by verifying the correctness of the received message.

4.2 The Protocol

Let $\{f_s\}_{s\in\{0,1\}^*}$ be a pseudorandom function, and let $[x]_s$ denote the message $(x, f_s(x))$ where $f_s(x)$ is the message authentication code of x with respect to s.

The MAP1 protocol is used to establish a mutual authentication between any two agents A and B among a set of agents $\mathbb{A}$ who share a key s. At the beginning, all agents share a pseudorandom function and a secret random element $s \in \{0,1\}^k$, where k is the security parameter. When agent A wants to communicate with agent B, A sends to B a random challenge (a nonce) $R_A \in_R \{0,1\}^k$. B responds by making up a random challenge $R_B \in_R \{0,1\}^k$ and returning $[b.a.R_A.R_B]_s$, where a and b are descriptions of the identity of agents A and B, respectively. Then, A checks that the message received from B is of the right form and that it is correctly tagged as coming from B. If it is, A sends B the message $[a.R_B]_s$ and accepts. B checks that the message from A is of the right form and that it is correctly tagged as coming from A. If it is, B accepts. Fig. 1 depicts how the MAP1 protocol works.

The definition of correctness proposed by Bellare and Rogaway in [3] is based on the concept of *matching conversation*. All agents communicate via an adversarial network E, controlled by a probabilistic polynomial time algorithm, that can block, delay and/or modify messages, and possibly create new messages. Two agents A and B have a matching conversation if the following conditions hold:

1. every message that A sends out, except possibly the last, is subsequently delivered to B, with the response to this message being returned to A as its own next message;

2. every message B receives was previously generated by A and each message that B sends out is subsequently delivered to A, with the response that this message generates being returned to B as its own next message.

The first condition states that when A (that plays as a sender or initiator agent) sends a message to B, the message is not modified or blocked by the adversary E (except for the last message) and the response of B is correctly delivered to A, without changing the messages order. The second condition is very similar to the first one, but it is based on B's point of view (B plays as a receiver or responder agent).

Given an adversary E (that does not know the secret key s shared by the agents), E breaks the MAP1 protocol if it completes a mutual authentication with some agent X persuading X that the other participant is another agent Y. This means that X completes the protocol without a matching conversation with Y. More formally, MAP1 is a secure mutual authentication protocol if

- for each pair of agents X and Y, if X and Y have a matching conversation, then both agents accept;

- for any probabilistic polynomial time adversary E, the probability that E induces an agent X to accept a communication with another agent Y without a matching conversation with Y is negligible.

E, during the attack, can play as initiator or responder, or even in both roles if it tries to break the MAP1 protocol interacting with several agents.

4.3 The Correctness Proof of Bellare-Rogaway

The original proof that MAP1 is a secure mutual authentication protocol can be found in Appendix A of [3]. The proof is split into two parts. First it is shown that the probability of breaking the protocol when the agents share a truly random function is negligible; then it is shown that an adversary E that successfully attacks the MAP1 protocol with a non-negligible probability can be turned into a distinguisher for a pseudorandom function.

The second step is rather standard in cryptography: the distinguisher is an algorithm that simulates the interaction between the adversary E and the agents and that queries the message authentication scheme whenever it simulates a real agent that computes a message authentication code. The distinguisher returns 1 whenever it successfully induces an agent A to accept without a matching conversation. The probability of returning 1 is then significantly different if the message authentication scheme is given by a truly random function or by a pseudorandom function. Though this construction is described in a semi-formal language, it is quite standard and widely accepted.

The first step is based on an explicit computation of the probability that the adversary induces acceptance without a matching conversation when the message authentication scheme is given by a truly random function. The short proof must be read with great attention because of the high number of potential pitfalls. It is a classical proof where we reason about global properties of computations by arguing back and forth about properties of different computational steps. These are typical arguments employed in correctness proofs for distributed and concurrent systems. In the specific case the argument is complicated further by the presence of probabilities. More or less the argument is a sequence of semi-formal statements about what messages are generated, in what order, who can have generated them (and with which probability), and whether messages can be repeated (and with which probability). Arguments about uniqueness of nonces and unforgeability of message authentication codes are intermixed. Our suggestion is that the use of polynomially accurate simulations in this context can provide us with the same simplifications that the simulation method provided in the area of distributed systems (cf. [14]).

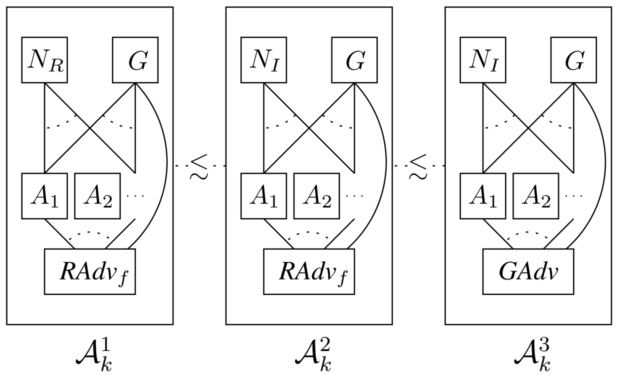

Figure 2. The three levels of abstraction for MAP1.

4.4 Our Correctness Proof

We now give an outline of the correctness proof of the MAP1 protocol based on polynomially accurate simulations. We describe the protocol at three levels of abstraction. The lowest level description consists of the actual agents that receive the secret s from a secret generator and receive nonces from a device that generates random numbers. The adversary is controlled by a generic probabilistic polynomial time algorithm. At the intermediate level nonces are generated by an ideal device that keeps track of what was distributed earlier, while at the highest level the adversary is purely nondeterministic and is not allowed to generate new message authentication codes without obtaining them from the agents. Figure 2 depicts the three levels of abstraction.

The highest level abstraction is similar in style to the Dolev-Yao model where we assume perfect cryptography, while the description in three levels is similar in style to the game transformations proposed in [4, 5, 25]. The most abstract system can be shown easily not to exhibit any attack by employing ordinary well known techniques for purely nondeterministic systems. The novel element here is the use of simulation relations to relate the three levels.

We exhibit a polynomially accurate simulation for each pair of neighbor abstractions, use transitivity to state that there is a polynomially accurate simulation from the lowest level to the highest level abstraction, use Theorem 1 to argue that the probability of low level computations that do not have corresponding high level computations is negligible, and use the fact that at the highest level there are no attacks to deduce that at the lowest level the probability of attack is negligible. The crucial and interesting point is that at each level the negation of the step condition is the negation of the key property of nonces or the definition of a successful forger for a message authentication scheme depending on the simulation relation we are analyzing.

Now we give a more detailed description of the three levels of the abstraction. The lowest level, depicted on the left of Figure 2, consists of several automata, each one parameterized by a security parameter k (we don't add such parameter to the automata names for clarity). The automaton G is a secret generator that generates and provides the agents with a secret s that is used as the key of the message authentication scheme of MAP1 protocol. The automaton N_R models a real nonce generator. Whenever an agent needs a nonce, it sends a request to N_R and obtains a random value taken from $\{0,1\}^k$ as answer. The set $\{A_1, A_2, \ldots\}$ is a numerable set of automata that describe end-points of sessions of the protocol. That is, each automaton A_i corresponds to some oracle $\Pi^t_{X,Y}$ of [3], where oracle $\Pi^t_{X,Y}$ describes the participant X trying to authenticate participant Y in session t, where t is different for each authentication attempt. Communication between agents and secret and nonce generators is private, while communication between agents is performed using a network that is controlled by the adversary *RAdv*$_f$. The network keeps an history variable that contains all previous messages sent and received by agents, which is used to select the next action to perform (e.g., delivering messages, casting new messages, blocking messages, ...). The choices of network should be computable in probabilistic polynomial time. For this reason, the adversary *RAdv*$_f$ is parameterized by a probabilistic polynomial time function f, so that the transition enabled from a state s is $f(s)$.

The intermediate level, depicted in the middle of Figure 2, differs from the lowest level only in the nonce generator automaton. N_I models an ideal nonce generator that guarantees that nonces are never repeated. This implies that unicity of nonces chosen by agents is guaranteed by definition.

The highest level, depicted in the right of Figure 2, differs from the intermediate level only in the automaton that controls the network. The new adversary, denoted by *GAdv*, is a nondeterministic automaton that is allowed to perform any action except for casting new message authentication codes without obtaining them from the agents. More precisely, we define a function *Not_Bad* that, given a secret s and a history *history*, returns the set of messages where all subparts that are tagged correctly with a message authentication code relative to s are taken from *history*. That is, no new correct tag is cast. Then we require *GAdv* to generate only those messages that are in the outcome of function *Not_Bad*. This implies that unforgeability of message authentication scheme is warranted by definition.

4.4.1 Automata Specification

We now provide the automata that describe the participants and adversaries of the MAP1 protocol. We adopt the notation used by Lynch in [14] (cf. Figure 3). Each automaton

```
$G^k(\mathbb{A})$
**Signature:**
Output:
    $secret^t_{X,Y}(s)$, $s \in \{0,1\}^k$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$
    $secret_A(s)$, $s \in \{0,1\}^k$

**State:**
    $value \in \{0,1\}^k$, initially $v \in_R \{0,1\}^k$

**Transitions:**

Output $secret^t_{X,Y}(s)$
    Precondition:
        $s = value$
    Effect:
        none

Output $secret_A(s)$
    Precondition:
        $s = value$
    Effect:
        none
```

Figure 3. The Secret Generator, G^k

```
$N^k_I(\mathbb{A}), N^k_R(\mathbb{A})$
**Signature:**
Input:
    $nonce\_request^t_{X,Y}$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$
Output:
    $nonce\_response^t_{X,Y}(n)$, $n \in \{0,1\}^k$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

**State:**
    $fresh\_nonces \subseteq \{0,1\}^k$, initially $\{0,1\}^k$
    $value^t_{X,Y} \in \{0,1\}^k \cup \{\bot\}$, initially $\bot$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

**Transitions:**

Input $nonce\_request^t_{X,Y}$
    Effect:
        $value^t_{X,Y} := \begin{cases} v \in_R fresh\_nonces & \text{for } N^k_I(\mathbb{A}) \\ v \in_R \{0,1\}^k & \text{for } N^k_R(\mathbb{A}) \end{cases}$
        $fresh\_nonces := fresh\_nonces \setminus \{v\}$

Output $nonce\_response^t_{X,Y}(n)$
    Precondition:
        $n = value^t_{X,Y}$
    Effect:
        $value^t_{X,Y} := \bot$
```

Figure 4. The Nonce Generators, $N^k_I(\mathbb{A})$ and $N^k_R(\mathbb{A})$

is described by three parts: signature, states, and transitions. The signature lists the actions of the automaton, partitioned into input, output, and internal. Each action has a name, a sequence of parameters, and a set of values each parameter may assume. The states are described by a set of variables. Each variable assumes values in a given set, and the start state is given by the initial value of each variable. Transitions specify, for each action, what is the effect of the action on the state, that is, how the state evolves. Output and internal actions have also a precondition that specifies when they are enabled. Input actions are assumed to be always enabled, and thus no precondition is specified for them. We use the symbol $\in$ to denote denote the fact that a value is chosen arbitrarily from some set, and we use the symbol $\in_R$ to denote the fact that a value is chosen randomly and uniformly from a finite set.

Figure 3 depicts the secret generator G. It starts with a secret s, chosen randomly in $\{0,1\}^k$, which is then sent to all agents via actions of the form *secret*$^t_{X,Y}$. The secret is sent also to the adversary via action *secret*$_A$, though the real adversary will discard the value received. The value of the secret will be used by the good adversary to prevent the generation of forged signatures.

Figure 4 shows the ideal and the real nonce generators N_I and N_R, respectively. The two automata are almost identical. Both automata keep a set *fresh_nonces* of fresh nonces, i.e., values that are not yet returned as nonces, which is is initialized to the set of all possible nonces, i.e., $\{0,1\}^k$. When the automaton receives an input *nonce_request*, it chooses a new nonce, removes it from the set of fresh nonces, and assigns it to a local variable to be used by the corresponding *nonce_response* action. The difference between the two automata is on how the new nonce is chosen: the ideal generator chooses it randomly in *fresh_nonces*, while the real nonce generator chooses it randomly in $\{0,1\}^k$. Thus N_R always return fresh nonces, while N_I may generate repeated nonces. Observe that variable *fresh_nonces* is not needed in N_R, but it is convenient to keep it to simplify the formulation of the simulation relations.

Figures 5 and 6 depict the $MAP1^t_{X,Y}$ automaton that describes an agent X trying to authenticate to another agent Y in session t. Agent X may play either as a sender or as a receiver, and the role of X is determined by the first input received by the automaton: if the first input is a *start_init* action, then X acts as sender (or initiator) agent; if the first input is a *receive1* action, then X acts as a receiver agent. The state of the automaton has two variables R_X, R_Y that store local copies of the nonces of X and Y, respectively; a variable *secret* that stores the secret key of the message authentication scheme; a variable *accept* that assumes value true when the automaton accepts the authentication; a *nonce_requested* variables that is used to remember when a nonce request is pending; and a program counter *pc* that keeps track of the current position in the flow of the MAP1 protocol. The automaton switches to an error state

```
$MAP1^{k,t}_{X,Y}$
**Signature:**
Input:
    $\mathit{start\_init}^t_{X,Y}$
    $\mathit{receive1}^t_{X,Y}(m)$, $m \in \{0,1\}^k$
    $\mathit{receive2}^t_{X,Y}(m)$, $m \in \{0,1\}^{5k}$
    $\mathit{receive3}^t_{X,Y}(m)$, $m \in \{0,1\}^{3k}$
    $\mathit{nonce\_response}^t_{X,Y}(n)$, $n \in \{0,1\}^k$
    $\mathit{secret}^t_{X,Y}(s)$, $s \in \{0,1\}^k$
Output:
    $\mathit{nonce\_request}^t_{X,Y}$
    $\mathit{send1}^t_{X,Y}(m)$, $m \in \{0,1\}^k$
    $\mathit{send2}^t_{X,Y}(m)$, $m \in \{0,1\}^{5k}$
    $\mathit{send3}^t_{X,Y}(m)$, $m \in \{0,1\}^{3k}$

**State:**
    $R_X, R_Y \in \{0,1\}^k \cup \{\bot\}$, initially $\bot$
    $\mathit{secret} \in \{0,1\}^k \cup \{\bot\}$, initially $\bot$
    $\mathit{pc} \in \{\mathtt{error}, \mathtt{end}, \mathtt{wait1}, \mathtt{wait2}, \mathtt{wait3},$
        $\mathtt{send1}, \mathtt{send2}, \mathtt{send3}\}$, initially $\mathtt{wait1}$
    $\mathit{nonce\_requested} \in \{T, F\}$, initially $F$
    $\mathit{accept} \in \{T, F\}$, initially $F$


**Transitions:**

Input $\mathit{secret}^t_{X,Y}(s)$
    Effect:
        $\mathit{secret} := s$

Output $\mathit{nonce\_request}^t_{X,Y}$
    Precondition:
        $\mathit{pc} \in \{\mathtt{send1}, \mathtt{send2}\} \wedge R_X = \bot \wedge \neg \mathit{nonce\_requested}$
    Effect:
        $\mathit{nonce\_requested} := T$

Input $\mathit{nonce\_response}^t_{X,Y}(n)$
    Effect:
        if $\neg \mathit{nonce\_requested}$ then
         $\mathit{pc} := \mathtt{error}$
        else
         $R_X := n$
         $\mathit{nonce\_requested} := F$
        fi

Input $\mathit{start\_init}^t_{X,Y}$
    Effect:
        if $\mathit{pc} = \mathtt{wait1}$ then
         $\mathit{pc} := \mathtt{send1}$
        else
         $\mathit{pc} := \mathtt{error}$
        fi
```

Figure 5. The MAP1 Agent, $MAP1^{k,t}_{X,Y}$, Part I

```
**Transitions:**

Output $\mathit{send1}^t_{X,Y}(m)$
    Precondition:
        $\mathit{pc} = \mathtt{send1} \wedge m = R_X \neq \bot \wedge \mathit{secret} \neq \bot$
    Effect:
        $\mathit{pc} := \mathtt{wait2}$

Input $\mathit{receive2}^t_{X,Y}(m)$
    Effect:
        if $\mathit{pc} = \mathtt{wait2} \wedge$
           $\exists r \in \{0,1\}^k . m = [y.x.R_X.r]_{\mathit{secret}}$ then
         $R_Y := r$
         $\mathit{pc} := \mathtt{send3}$
        else
         $\mathit{pc} := \mathtt{error}$
        fi

Output $\mathit{send3}^t_{X,Y}(m)$
    Precondition:
        $\mathit{pc} = \mathtt{send3} \wedge m = [x.R_Y]_{\mathit{secret}}$
    Effect:
        $\mathit{pc} := \mathtt{end}$
        $\mathit{accept} := T$

Input $\mathit{receive1}^t_{X,Y}(m)$
    Effect:
        if $\mathit{pc} = \mathtt{wait1}$ then
         $\mathit{pc} := \mathtt{send2}$
         $R_Y := m$
        else
         $\mathit{pc} := \mathtt{error}$
        fi

Output $\mathit{send2}^t_{X,Y}(m)$
    Precondition:
        $\mathit{pc} = \mathtt{send2} \wedge R_X \neq \bot \wedge \mathit{secret} \neq \bot \wedge$
            $m = [x.y.R_Y.R_X]_{\mathit{secret}}$
    Effect:
        $\mathit{pc} := \mathtt{wait3}$

Input $\mathit{receive3}^t_{X,Y}(m)$
    Effect:
        if $\mathit{pc} = \mathtt{wait3} \wedge m = [y.R_X]_{\mathit{secret}}$ then
         $\mathit{pc} := \mathtt{end}$
         $\mathit{accept} := T$
        else
         $\mathit{pc} := \mathtt{error}$
        fi
```

Figure 6. The MAP1 Agent, $MAP1^{k,t}_{X,Y}$, Part II

($pc = \mathtt{error}$) as soon as an unexpected input or a badly formatted message is received. From the error state the automaton does not perform any output action and ignores the effects of all input actions. The sequence of actions follows the MAP1 protocol as proposed in [3].

Figures 7 and 8 show the good adversary. First of all the adversary waits for the secret from the secret generator G. Then it alternates internal generation of messages according to function *Not_Bad*, which guarantees no forging of signatures, and delivery of messages to agents. All inputs from the agents are simply added to the history.

Figures 9 and 10 depict the real adversary. Also in this case the adversary waits for the secret from G, but the actual value of the secret is discarded. After that, the adversary behaves sequentially: it generates internally a new message, including the destination, according to a probabilistic poly-

$\mathit{GAdv}^k(\mathbb{A})$

Signature:

Input:

$\mathit{secret}_A(s)$, $s \in \{0,1\}^k$

$\mathit{send1}^t_{X,Y}(m)$, $m \in \{0,1\}^k$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

$\mathit{send2}^t_{X,Y}(m)$, $m \in \{0,1\}^{5k}$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

$\mathit{send3}^t_{X,Y}(m)$, $m \in \{0,1\}^{3k}$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

Output:

$\mathit{start\_init}^t_{X,Y}$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

$\mathit{receive1}^t_{X,Y}(m)$, $m \in \{0,1\}^k$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

$\mathit{receive2}^t_{X,Y}(m)$, $m \in \{0,1\}^{5k}$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

$\mathit{receive3}^t_{X,Y}(m)$, $m \in \{0,1\}^{3k}$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

Internal:

create_message

State:

$\mathit{history} \in \mathtt{Sequences}(\mathit{Actions}(\mathbb{A}) \times M)$, initially $\emptyset$,
$M = \{0,1\}^k \cup \{0,1\}^{3k} \cup \{0,1\}^{5k}$

$\mathit{message} \in \{0,1\}^k \cup \{0,1\}^{3k} \cup \{0,1\}^{5k}$

$\mathit{secret} \in \{0,1\}^k \cup \{\perp\}$, initially $\perp$

Transitions:

```
Input  $\mathit{secret}_A(s)$
    Effect:
        $\mathit{secret} := s$

Internal  create_message
    Precondition:
        $\mathit{secret} \neq \perp$
    Effect:
        $\mathit{message} := m \in \mathit{Not\_Bad}(\mathit{secret}, \mathit{history})$
```

Figure 7. The Good Adversary, $\mathit{GAdv}^k(\mathbb{A})$, Part I

Transitions:

```
Output  $\mathit{start\_init}^t_{X,Y}$
    Precondition:
        $\mathit{secret} \neq \perp$
    Effect:
        $\mathit{history} := \mathit{history} \vdash (\mathit{start\_init}^t_{X,Y}, \mathit{message})$

Input  $\mathit{send1}^t_{X,Y}(m)$
    Effect:
        $\mathit{history} := \mathit{history} \vdash (\mathit{send1}^t_{X,Y}, m)$

Output  $\mathit{receive1}^t_{X,Y}(m)$
    Precondition:
        $m = \mathit{message}$
    Effect:
        $\mathit{history} := \mathit{history} \vdash (\mathit{receive1}^t_{X,Y}, m)$

Input  $\mathit{send2}^t_{X,Y}(m)$
    Effect:
        $\mathit{history} := \mathit{history} \vdash (\mathit{send2}^t_{X,Y}, m)$

Output  $\mathit{receive2}^t_{X,Y}(m)$
    Precondition:
        $m = \mathit{message}$
    Effect:
        $\mathit{history} := \mathit{history} \vdash (\mathit{receive2}^t_{X,Y}, m)$

Input  $\mathit{send3}^t_{X,Y}(m)$
    Effect:
        $\mathit{history} := \mathit{history} \vdash (\mathit{send3}^t_{X,Y}, m)$

Output  $\mathit{receive3}^t_{X,Y}(m)$
    Precondition:
        $m = \mathit{message}$
    Effect:
        $\mathit{history} := \mathit{history} \vdash (\mathit{receive3}^t_{X,Y}, m)$
```

Figure 8. The Good Adversary, $\mathit{GAdv}^k(\mathbb{A})$, Part II

nomial time function f, it forwards the generated message to the chosen destination, and, if specified in the MAP1 protocol, waits for the answer. Then the cycle is repeated. The correctness of the cycle is guaranteed by a boolean variable *enable_action_creation*, which is true only when a new message can be generated.

4.4.2 Some Considerations on the Automata

We have been very careful in the definition of the real adversary, and in particular we have ensured that its behavior is sequential. One reason for doing this is that in the definition of correct message authentication schemas the forger is a sequential process, and thus, if we want the negation of the step condition to become the definition of a forger, we need to make sure that we will deal with a sequential process.

It would be desirable to be able to reason with a more general, non-sequential, adversary, but unfortunately it is not possible to do it in the current setting. Suppose we allow the real adversary to generate messages according to f in any order, without necessarily waiting for the answers from the agents. Then we can build a scheduler, and an appropriate function f, where the adversary initializes k sessions of the MAP1 protocol, say $S_1, ..., S_k$, and make sure that session S_i responds only if the i^{th} bit of the secret is 1. In this way the adversary knows the value of the secret and is therefore able sign messages. In other words we can resolve nondeterminism to create a covert channel that communicates the secret to the adversary. Solutions to this problem are studied already in the literature [8,18,21] and it is worth investigating how polynomially accurate simulations can be adapted to such frameworks. Here we have chosen to avoid restrictions to the schedulers to keep the treatment simple and to show that it is also possible to remove dangerous nondeterminism and work with unrestricted schedulers.

Another observation about our definition of the adver-

$\mathit{RAdv}_f^k(\mathbb{A})$

Signature:

Input:

$\mathit{secret}_A(s)$, $s \in \{0,1\}^k$
$\mathit{send1}_{X,Y}^t(m)$, $m \in \{0,1\}^k$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$
$\mathit{send2}_{X,Y}^t(m)$, $m \in \{0,1\}^{5k}$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$
$\mathit{send3}_{X,Y}^t(m)$, $m \in \{0,1\}^{3k}$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

Output:

$\mathit{start\_init}_{X,Y}^t$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$
$\mathit{receive1}_{X,Y}^t(m)$, $m \in \{0,1\}^k$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$
$\mathit{receive2}_{X,Y}^t(m)$, $m \in \{0,1\}^{5k}$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$
$\mathit{receive3}_{X,Y}^t(m)$, $m \in \{0,1\}^{3k}$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

Internal:

create_action

State:

$\mathit{history} \in \mathtt{Sequences}(\mathit{Actions}(\mathbb{A}) \times M)$, initially $\emptyset$,
$M = \{0,1\}^k \cup \{0,1\}^{3k} \cup \{0,1\}^{5k}$
$\mathit{action} \in \mathit{Actions}(\mathbb{A}) \cup \{\bot\}$, initially $\bot$
$\mathit{message} \in \{0,1\}^k \cup \{0,1\}^{3k} \cup \{0,1\}^{5k}$
$\mathit{enable\_action\_creation} \in \{T, F\}$, initially T
$\mathit{run\_enabled} \in \{T, F\}$, initially F

Transitions:

```
Input $\mathit{secret}_A(s)$
    Effect:
        $\mathit{run\_enabled} := T$

Internal $\mathit{create\_action}$
    Precondition:
        $\mathit{run\_enabled} \wedge \mathit{enable\_action\_creation}$
    Effect:
        $\mathit{enable\_action\_creation} := F$
        $(\mathit{action}, \mathit{message}) := f(\mathit{history})$
```

Figure 9. The Real Adversary, $\mathit{RAdv}_f^k(\mathbb{A})$, Part I

saries is that we have separated message generation from message delivery. We could easily avoid this separation, but in this case we would be forced to use probabilistic automata with generative transitions [23] to describe the real adversary, which have a more complex theory. Once again, our choice is to keep the presentation simple and focus on the ideas behind polynomially accurate simulations. Finally, we could easily remove the internal generation of messages from the good adversary and yet avoid generative transitions. However, in this case we would need to use weak simulations to deal with the internal moves of the real adversary. In this paper we have chosen to focus only on strong relations, again for the sake of simplicity.

4.4.3 The Correctness Proof

Now we are ready to show the correctness of the MAP1 protocol.

Transitions:

```
Output $\mathit{start\_init}_{X,Y}^t$
    Precondition:
        $\mathit{action} = \mathit{start\_init}_{X,Y}^t$
    Effect:
        $\mathit{action} := \bot$
        $\mathit{history} := \mathit{history} \vdash (\mathit{start\_init}_{X,Y}^t, \mathit{message})$

Input $\mathit{send1}_{X,Y}^t(m)$
    Effect:
        $\mathit{history} := \mathit{history} \vdash (\mathit{send1}_{X,Y}^t, m)$
        $\mathit{enable\_action\_creation} := T$

Output $\mathit{receive1}_{X,Y}^t(m)$
    Precondition:
        $\mathit{action} = \mathit{receive1}_{X,Y}^t \wedge m = \mathit{message}$
    Effect:
        $\mathit{history} := \mathit{history} \vdash (\mathit{receive1}_{X,Y}^t, m)$
        $\mathit{action} := \bot$

Input $\mathit{send2}_{X,Y}^t(m)$
    Effect:
        $\mathit{history} := \mathit{history} \vdash (\mathit{send2}_{X,Y}^t, m)$
        $\mathit{enable\_action\_creation} := T$

Output $\mathit{receive2}_{X,Y}^t(m)$
    Precondition:
        $\mathit{action} = \mathit{receive2}_{X,Y}^t \wedge m = \mathit{message}$
    Effect:
        $\mathit{history} := \mathit{history} \vdash (\mathit{receive2}_{X,Y}^t, m)$
        $\mathit{action} := \bot$

Input $\mathit{send3}_{X,Y}^t(m)$
    Effect:
        $\mathit{history} := \mathit{history} \vdash (\mathit{send3}_{X,Y}^t, m)$
        $\mathit{enable\_action\_creation} := T$

Output $\mathit{receive3}_{X,Y}^t(m)$
    Precondition:
        $\mathit{action} = \mathit{receive3}_{X,Y}^t \wedge m = \mathit{message}$
    Effect:
        $\mathit{history} := \mathit{history} \vdash (\mathit{receive3}_{X,Y}^t, m)$
        $\mathit{enable\_action\_creation} := T$
        $\mathit{action} := \bot$
```

Figure 10. The Real Adversary, $\mathit{RAdv}_f^k(\mathbb{A})$, Part II

Proposition 6. $\{\mathcal{A}_k^1\}_{k \in K} \lesssim \{\mathcal{A}_k^2\}_{k \in K}$

Proof. Define the relation family $\{\mathcal{R}_k\}_{k \in K}$ as the family of identity relations.

The condition on start states is trivially true. Suppose that the step condition does not hold. This means that there exist $c \in \mathbb{N}$, $p \in \mathit{Poly}$ such that for all $\bar{k} \in \mathbb{N}$ there exist $k > \bar{k}$, ν_1, ν_2, γ and η_1 such that ν_1 is reached by A_k^1 within $p(k)$ steps, $\nu_1\ \mathcal{L}(\mathcal{R}_k, \gamma)\ \nu_2$, $\nu_1 \longrightarrow \eta_1$ and there is no η_2 such that $\nu_2 \longrightarrow \eta_2$ and $\eta_1\ \mathcal{L}(\mathcal{R}_k, \gamma + k^{-c})\ \eta_2$.

Figure 11 gives a graphical representation of the transi-

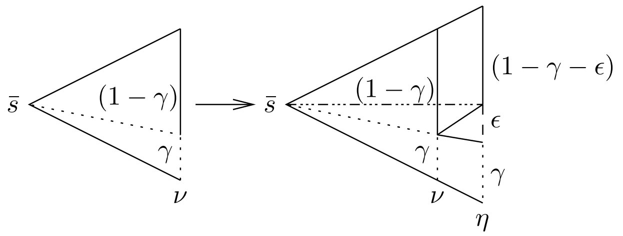

Figure 11. Graphical representation of the transition $\nu \longrightarrow \eta$.

tion $\nu_1 \longrightarrow \eta_1$ where ε represents the part of the transition that can not be emulated from ν_2; hence $\varepsilon > k^{-c}$.

Since $\mathcal{A}_k^1$, $\mathcal{A}_k^2$ differ only in the nonce generators, and since N_I may return any nonce except for repeated nonces, the ε fraction above corresponds to generation of repeated nonces. That is, the right side of Figure 11 represents a computation of $\mathcal{A}_k^1$ of length at most $p(k)+1$ where a repeated nonce is generated with probability at least k^{-c}. Summing up, there exist $c \in \mathbb{N}$, $p \in$ *Poly* such that for all $\bar{k} \in \mathbb{N}$ there exist $k > \bar{k}$ and η_1 such that η_1 has length at most $p(k)+1$ and the probability of repeated nonces in η_1 is at least k^{-c}. This contradicts the fact that N_R is a real nonce generator since the statement above is the negation of Claim 1 after using a polynomial $p'(k)$ to denote $p(k)+1$ and observing that at most $p'(k)$ nonces are generated within $p'(k)$ steps. □

Proposition 7. $\{\mathcal{A}_k^2\}_{k\in K} \lesssim \{\mathcal{A}_k^3\}_{k\in K}$

Proof. Define the relation family $\{\mathcal{R}_k\}_{k\in K}$ as the family of identity relations.

The condition on start states is trivially true. Suppose that step condition does not hold. This means that there exist $c \in \mathbb{N}$, $p \in$ *Poly* such that for all $\bar{k} \in \mathbb{N}$ there exist $k > \bar{k}$, ν_2, ν_3, γ and η_2 such that ν_2 is reached by A_k^2 within $p(k)$ steps, $\nu_2\ \mathcal{L}(\mathcal{R}_k,\gamma)\ \nu_3$, $\nu_2 \longrightarrow \eta_2$ and there is no η_3 such that $\nu_3 \longrightarrow \eta_3$ and $\eta_2\ \mathcal{L}(\mathcal{R}_k, \gamma + k^{-c})\ \eta_3$.

Figure 11 gives a graphical representation of the transition $\nu_2 \longrightarrow \eta_2$ where ε represents the part of the transition that can not be emulated from ν_3; hence $\varepsilon > k^{-c}$.

Since $\mathcal{A}_k^2$, $\mathcal{A}_k^3$ differ only in the adversaries, and since *GAdv* may perform any action except for casting new message authentication codes without obtaining them from the agents, the ε fraction above corresponds to generation of new message authentication codes. That is, the right side of Figure 11 represents a computation of $\mathcal{A}_k^2$ of length at most $p(k)+1$ where a new message authentication code is generated with probability at least k^{-c}. Summing up, there exist $c \in \mathbb{N}$, $p \in$ *Poly* such that for all $\bar{k} \in \mathbb{N}$ there exist $k > \bar{k}$ and η_2 such that η_2 has length at most $p(k)+1$ and the probability to generate new message authentication codes in η_2 is at least k^{-c}. This contradicts the fact that the message authentication scheme used by MAP1 is a secure message authentication scheme since the statement above is the negation of the negligible probability of successful forger after using a polynomial $p'(k)$ to denote $p(k)+1$ and observing that at most $p'(k)$ message authentication codes are requested within $p'(k)$ steps. □

As we can see, in both proofs the negation of the step condition leads to a negation of properties of underlying cryptographic primitives. In the first proof we have negated the unicity of nonces, in the second we have negated the security of the used message authentication scheme.

5 Conclusion

We have proposed a new notion of polynomially accurate simulation relation for probabilistic automata with the aim of adapting the simulation method to the analysis of protocols that involve cryptographic elements. The new simulation relations permit transitions to be matched up to some error that is bounded by any polynomial provided that computations are of polynomial length. In this way, the global property of matching polynomial computations up to some negligible probability can be reduced to a local property of single transitions.

We have applied our simulation relations to a simple case study taken from the literature on cryptography, showing how the failure of the step condition can be turned immediately into the definition of an attack against an underlying cryptographic primitive. In this way we obtain proofs that are more rigorous than those based on semi-formal arguments.

We expect that the potentials of polynomially accurate simulations illustrated above can have positive impacts on several other scenarios. In particular, we are interested in studying how our approach may be beneficial to the work in [2, 4, 5, 25] and how polynomially accurate simulations can be used to express more precisely the connections between the symbolic and computational models of security.

The definitions given in this paper do not distinguish between visible and invisible actions, that is, they are *strong* simulations. It is not difficult to extend our definitions to the weak case; however, we have some degrees of freedom in imposing conditions on the complexity of matching transitions, and we prefer to use more case studies as guidelines for choosing the most adequate definition.

In the case study of this paper we have been very careful on the use of nondeterminism so that unrestricted schedulers do not have the possibility to create undesired covert channels. Another line of research prevents covert channels by restricting the power of nondeterminism [8, 18, 21]. It is worth investigating how polynomially accurate simulations can be adapted to such frameworks, and in particular whether polynomially accurate simulations could be used as a sound proof technique for the approximated language inclusion relations of [8].

References

[1] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP TCS*, *LNCS* 2000, pages 3–22. Springer, 2000.

[2] M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015, 2003.

[3] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO '93*, pages 232–249. Springer, 1994.

[4] B. Blanchet. A computationally sound mechanized prover for security protocols. Cryptology ePrint Archive, Report 2005/401, 2005.

[5] B. Blanchet and D. Pointcheval. Automated security proofs with sequences of games. Cryptology ePrint Archive, Report 2006/069, 2006.

[6] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *The 42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, 2001.

[7] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Using Probabilistic I/O Automata to analyze an oblivious transfer protocol. Technical Report 2005/452, Cryptology ePrint Archive, 2005.

[8] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Time-bounded task-PIOAs: A framework for analyzing security protocols. In *DISC 2006*, *LNCS* 4167, pages 238–253. Springer, 2006.

[9] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[10] O. Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, 2001.

[11] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

[12] R. Janvier, Y. Lakhnech, and L. Mazare. Completing the picture: Soundness of formal encryption in the presence of active adversaries. Technical Report TR-2004-19, Verimag, 2005.

[13] P. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security analysis. In *FM '99*, *LNCS* 1708, pages 776–793. Springer, 1999.

[14] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[15] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *PODC '87*, pages 137–151. ACM Press, 1987.

[16] D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 2004.

[17] R. Milner. *Communication and Concurrency*. Prentice-Hall International, Englewood Cleiffs, 1989.

[18] J. C. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theoretical Computer Science*, 353(1):118–164, 2006.

[19] S. Mitra and N. Lynch. Approximate simulations for task-structured Probabilistic I/O Automata. In *PAuL06*, 2006.

[20] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *CCS '00*, pages 245–254. ACM Press, 2000.

[21] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *SP '01*, pages 184–200. IEEE Computer Society, 2001.

[22] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1995. Also appears as technical report MIT/LCS/TR-676.

[23] R. Segala. Probability and nondeterminism in operational models of concurrency. In *CONCUR 2006 - Concurrency Theory*, *LNCS* 4137, pages 64–78. Springer, 2006.

[24] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[25] V. Shoup. A tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004.

[26] A. Sokolova and E. P. de Vink. Probabilistic automata: System types, parallel composition and comparison. In *Validation of Stochastic Systems: A Guide to Current Research*, *LNCS* 2925, pages 1–43. Springer, 2004.