Composable Dynamic Secure Emulation

Pierre Civit
pierre.civit@lip6.fr
Sorbonne Université, CNRS, LIP6
Paris, France

Maria Potop-Butucaru*
maria.potop-butucaru@lip6.fr
Sorbonne Université, CNRS, LIP6
Paris, France

ABSTRACT

This work extends the *composable secure-emulation* of Canetti et al. [4] to dynamic settings. Our work builds on top of dynamic probabilistic I/O automata, a recent framework introduced to model dynamic probabilistic systems. Our extension is an important tool towards the formal verification of protocols combining probabilistic distributed systems and cryptography in dynamic settings (e.g. blockchains, secure distributed computation, cybersecure distributed protocols etc).

1 INTRODUCTION

Blockchains or distributed ledger technologies are a game changer for distributed computing area. Blockchains are an evolved form of the distributed computing concept of replicated state machine, in which multiple agents see the evolution of a state machine in a consistent form. At the core of both mechanisms there are distributed computing fundamental elements (e.g. communication primitives and semantics, consensus algorithms, and consistency models) and also sophisticated cryptographic tools. Recently, [9] stated that despite the tremendous interest related to distributed ledgers, no formal abstraction of these objects has been proposed. In particular it was stated that there is a need for the formalization of the distributed systems that are at the heart of most cryptocurrency implementations. Therefore, an extremely important aspect of blockchain foundations is a proper model for the entities involved and their potential behavior. The formalisation of blockchain systems has to combine models of underlying distributed and cryptographic building blocks under the same hood.

The formalisation of distributed systems has been pioneered by Lynch and Tuttle [12]. They proposed the formalism of *Input/Output Automata* to model deterministic distributed system. Later, this formalism is extended by Segala [14] to *Probabilistic Input/Output Automata* in order to model randomized distributed systems. In order to analyze cryptographic protocols Canetti & al. [3] extends this framework to *task-structured probabilistic Input/Output automata* (TPIOA). Task-structured probabilistic Input/Output automata are Probabilistic Input/Output automata extended with tasks that are equivalence classes on the set of actions. The task-structure allows a generalisation of "off-line scheduling" where the non-determinism of the system is resolved in advance by a *task-scheduler*, i. e. a sequence of tasks chosen in advance that trigger the actions among the enabled ones. They also define the parallel composition for this type of automata. Inspired by the literature in security area they define the notion of implementation for TPIOA. Furthermore, they provide compositional results for the implementation relation. Even thought the formalism proposed in [3] has been already used in the verification of various cryptographic protocols this formalism does not capture the dynamicity in blockchains systems such as Bitcoin or Ethereum where the set of participants dynamically changes. Moreover, this formalism does not cover blockchain systems where subchains can be created or destroyed at run time [13].

Interestingly, the model of dynamic behavior in distributed systems is an issue that has been addressed even before the born of blockchain systems. The increase of dynamic behavior in various distributed applications such as mobile agents and robots motivated the *Dynamic Input Output Automata* formalism introduced by Attie & Lynch in [1]. This formalism extends the *Input/Output Automata* formalism with the ability to change their signature dynamically (i.e. the set of actions in which the automaton can participate) and to create other I/O automata or destroy existing I/O automata. The formalism introduced in [1] does not cover the case of probabilistic distributed systems neither cryptographical aspects and therefore cannot be used in the verification of blockchains such as Algorand [6].

In order to cope with dynamicity and probabilistic nature of recent emergent systems (e.g. mobile probabilistic robots or probabilistic peer-to-peer protocols) the authors of [7] proposed an extension of the frameworks introduced in [3] and [1]. Their extension refined the definition of probabilistic configuration automata in order to cope with dynamic actions. The main result of this formalism is as follows: the implementation of probabilistic configuration automata is monotonic to automata creation and destruction. Although the framework proposed by [7] covers dynamic probabilistic protocols this framework cannot be used in order to model blockchain technologies since it does not cover cryptographic aspects of these protocols.

Our contribution. The current work builds on top of dynamic probabilistic I/O automata described in detail in [7] and proposes an extension of the *composable secure-emulation* of Canetti et al. [4] to dynamic settings. Our framework, composable dynamic secure emulation, enjoys the composability properties of secure-emulation of [4] face to dynamic behavior of underlying probabilistic I/O automata layer.

Paper organization. The paper is organized as follow. Section 2 recalls the key notions of the dynamic probabilistic I/O automata framework defined in [7]. Section 3 introduces the notion of scheduler and external perception. Section 4 extends the approximate implementation relation defined in [4] to dynamic settings. Furthermore, in this section we present the main contribution of the paper: the extension of the secure-emulation introduced in [2] to distributed systems and distributed scheduling copying with dynamic creation of automata. Moreover, this relation is shown to be composable.

2 BRACKGROUND ON PROBABILISTIC DYNAMIC INPUT/OUTPUT AUTOMATA

This section recalls the formalism defined in [7] i. e. probabilistic dynamic Input/Output Automata. The key components of this

framework are: probabilistic signature input/output automata, the notions of scheduler and implementation and the probabilistic configuration automata.

2.1 Preliminaries on probability and measure

A measurable space is denoted by $(S, \mathcal{F}_S)$, where S is a set and $\mathcal{F}_S$ is a σ-algebra over S. A measure space is denoted by $(S, \mathcal{F}_S, \eta)$ where η is a measure on $(S, \mathcal{F}_S)$. The product measure space $(S_1, \mathcal{F}_{S_1}, \eta_1) \otimes (S_2, \mathcal{F}_{S_2}, \eta_2)$ is the measure space $(S_1 \times S_2, \mathcal{F}_{S_1} \otimes \mathcal{F}_{S_2}, \eta_1 \otimes \eta_2)$, where $\mathcal{F}_{S_1} \otimes \mathcal{F}_{S_2}$ is the smallest σ-algebra generated by sets of the form $\{A \times B | A \in \mathcal{F}_{S_1}, B \in \mathcal{F}_{S_2}\}$ and $\eta_1 \otimes \eta_2$ is the unique measure s. t. for every $C_1 \in \mathcal{F}_{S_1}, C_2 \in \mathcal{F}_{S_2}$, $\eta_1 \otimes \eta_2(C_1 \times C_2) = \eta_1(C_1)\eta_2(C_2)$.

A discrete probability measure on a set S is a probability measure η on $(S, 2^S)$, such that, for each $C \subset S, \eta(C) = \sum_{c \in C} \eta(\{c\})$. We define $Disc(S)$ to be, the set of discrete probability measures on S. In the sequel, we often omit the set notation when we denote the measure of a singleton set. For a discrete probability measure η on a set S, $supp(\eta)$ denotes the support of η, that is, the set of elements $s \in S$ such that $\eta(s) \neq 0$. Given set S and a subset $C \subset S$, the Dirac measure δ_C is the discrete probability measure on S that assigns probability 1 to C. For each element $s \in S$, we note δ_s for $\delta_{\{s\}}$.

2.2 Probabilistic signature input/output automata (PSIOA)

A probabilistic signature input/output automata (PSIOA) is the result of the generalization of probabilistic input/output automata (PIOA) [14] and signature input/output automata (SIOA) [1]. A PSIOA is thus an automaton that can randomly move from one *state* to another in response to some *actions*. The set of possible actions is the *signature* of the automaton and is partitioned into *input*, *output* and *internal* actions.

We use the signature approach from [1]. We assume the existence of a countable set $Autids$ of unique probabilistic signature input/output automata (PSIOA) identifiers, an underlying universal set $Auts$ of PSIOA, and a mapping $aut : Autids \rightarrow Auts$. $aut(\mathcal{A})$ is the PSIOA with identifier $\mathcal{A}$. We use "the automaton $\mathcal{A}$" to mean "the PSIOA with identifier $\mathcal{A}$". We use the letters $\mathcal{A}, \mathcal{B}$, possibly subscripted or primed, for PSIOA identifiers. The executable actions of a PSIOA $\mathcal{A}$ are drawn from a signature $sig(\mathcal{A})(q) = (in(\mathcal{A})(q), out(\mathcal{A})(q), int(\mathcal{A})(q))$, called the state signature, which is a function of the current state q of $\mathcal{A}$, $in(\mathcal{A})(q)$, $out(\mathcal{A})(q)$, $int(\mathcal{A})$ (q) are pairwise disjoint sets of input, output, and internal actions, respectively. We aslo define $\widehat{sig}(\mathcal{A}) : q \in Q \mapsto \widehat{sig}(\mathcal{A})(q) = in(\mathcal{A})(q) \cup out(\mathcal{A})(q) \cup int(\mathcal{A})(q)$.

Definition 2.1 (PSIOA). A PSIOA $\mathcal{A} = (Q_{\mathcal{A}}, \bar{q}_{\mathcal{A}}, sig(\mathcal{A}), D_{\mathcal{A}})$, where:

- $Q_{\mathcal{A}}$ (a.k.a. $states(\mathcal{A})$) is a countable set of *states*, $(Q_{\mathcal{A}}, 2^{Q_{\mathcal{A}}})$ is a measurable space called the *state space*,
- $\bar{q}_{\mathcal{A}} \in Q_{\mathcal{A}}$ (a. k. a. $start(\mathcal{A})$) is the unique *start state*.
- $sig(\mathcal{A})$ is the signature function that maps each state $q \in Q_{\mathcal{A}}$ to a triplet $sig(\mathcal{A})(q) = (in(A)(q), out(A)(q), int(A)(q))$ of mutually disjoint countable set of *actions*, respectively called *input*, *output* and *internal* actions.
- $D_{\mathcal{A}} \subset Q_{\mathcal{A}} \times acts(\mathcal{A}) \times Disc(Q_{\mathcal{A}})$ ($D_{\mathcal{A}}$ a. k. a. $dtrans(\mathcal{A})$) is the set of *probabilistic discrete transitions* where $\forall (q, a, \eta) \in D_{\mathcal{A}} : a \in \widehat{sig}(\mathcal{A})(q)$.

In addition $\mathcal{A}$ must satisfy the following conditions [1]: $\forall q \in Q_{\mathcal{A}} : \forall a \in \widehat{sig}(\mathcal{A})(q), \exists! \eta_{(\mathcal{A},q,a)} \in Disc(Q_{\mathcal{A}}) : (q, a, \eta_{(\mathcal{A},q,a)}) \in D_{\mathcal{A}}$.

Notation. For every PSIOA $\mathcal{A} \triangleq (Q_{\mathcal{A}}, \bar{q}_{\mathcal{A}}, sig(\mathcal{A}), D_{\mathcal{A}})$, we note $states(\mathcal{A}) \triangleq Q_{\mathcal{A}}, start(\mathcal{A}) \triangleq \bar{q}_{\mathcal{A}}, dtrans(\mathcal{A}) \triangleq D_{\mathcal{A}}$. We also note $steps(\mathcal{A}) \triangleq \{(q, a, q') \in Q_{\mathcal{A}} \times acts(\mathcal{A}) \times Q_{\mathcal{A}} | \exists (q, a, \eta) \in D_{\mathcal{A}}, q' \in supp(\eta)\}$. Also we define $acts(\mathcal{A}) = \bigcup_{q \in Q} \widehat{sig}(\mathcal{A})(q)$, the "universal" set of all actions that $\mathcal{A}$ could possibly trigger. Moreover, for every mapping $m_{\mathcal{A}}$ with $states(\mathcal{A})$ as domain, we note $\widetilde{m}_{\mathcal{A}} = \bigcup_{q \in Q} m_{\mathcal{A}}(q)$. Finally for every pair of mapping $(m^1_{\mathcal{A}_1}, m^2_{\mathcal{A}_2})$ with $states(\mathcal{A}_1)$ and $states(\mathcal{A}_2)$ as respective domain, we note $m^1_{\mathcal{A}_1} \cup m^2_{\mathcal{A}_2} : (q_1, q_2) \in states(\mathcal{A}_1) \times states(\mathcal{A}_2) \mapsto m^1_{\mathcal{A}_1}(q_1) \cup m^2_{\mathcal{A}_2}(q_2)$

Definition 2.2 (fragment, execution and trace of PSIOA). An *execution fragment* of a PSIOA $\mathcal{A} = (Q_{\mathcal{A}}, \bar{q}_{\mathcal{A}}, sig(\mathcal{A}), D_{\mathcal{A}})$ is a finite or infinite sequence $\alpha = q^0 a^1 q^1 a^2 ...$ of alternating states and actions, such that:

(1) If α is finite, it ends with a state.
(2) For every non-final state q_i, $(q^i, a^i, q^{i+1}) \in steps(\mathcal{A})$

We write $fstate(\alpha)$ for q^0 (the first state of α), and if α is finite, we write $lstate(\alpha)$ for its last state. he length $|\alpha|$ of a finite execution fragment α is the number of transitions along α. We use $Frags(\mathcal{A})$ (resp., $Frags^*(\mathcal{A})$) to denote the set of all (resp., all finite) execution fragments of $\mathcal{A}$. An *execution* of $\mathcal{A}$ is an execution fragment α with $fstate(\alpha) = \bar{q}$. $Execs(\mathcal{A})$ (resp., $Execs^*(\mathcal{A})$) denotes the set of all (resp., all finite) executions of $\mathcal{A}$. The *trace* of an execution fragment α, written $trace(\alpha)$, is the restriction of α to the external actions of $\mathcal{A}$. We say that β is a trace of $\mathcal{A}$ if there is $\alpha \in Execs(\mathcal{A})$ with $\beta = trace(\alpha)$. A state $q \in Q_{\mathcal{A}}$ is said *reachable* if it exists a finite execution that ends with q. We note $reachable(\mathcal{A})$ the set of reachable states of $\mathcal{A}$. We define a concatenation operator $\frown$ for execution fragments as follows. If $\alpha = q^0 a^1 q^1 ... a^n q^n \in Frags^*(\mathcal{A})$ and $\alpha' = s^0 b^1 s^1 ... \in Frags(\mathcal{A})$, we define $\alpha \frown \alpha' = q^0 a^1 q^1 ... a^n s^0 b^1 s^1 ...$ only if $s^0 = q^n$, otherwise $\alpha \frown \alpha'$ is undefined. Hence the notation $\alpha \frown s^0 b^1 s^1 ...$ implicitly means $s^0 = lstate(\alpha)$. We also note $\alpha \frown (b^1, s^1)$ to states $\alpha \frown lstate(\alpha) b^1 s^1$. Let $\alpha, \alpha' \in Frags(\mathcal{A})$, then α is a proper prefix of α' iff $\exists \alpha'' \in Frags(\mathcal{A})$ such that $\alpha' = \alpha \frown \alpha''$ with $\alpha \neq \alpha'$. In that case, we note $\alpha < \alpha'$. We note $\alpha \leq \alpha'$ if $\alpha < \alpha'$ or $\alpha = \alpha'$ and say that α is a prefix of α'.

2.3 Signatures compatibility and composition

Tha main aim of IO formalism is to compose several automata $\mathbf{A} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$ and provide guarantees by composing the guarantees of the different elements of the system. Some syntaxic rules have to be satisfied before defining the composition operation.

Definition 2.3 (Compatible signatures). Let S be a set of signatures. Then S is compatible iff, $\forall sig, sig' \in S$, where $sig = (in, out, int)$, $sig' = (in', out', int')$ and $sig \neq sig'$, we have: 1. $(in \cup out \cup int) \cap int' = \emptyset$, and 2. $out \cap out' = \emptyset$.

[1] The condition could allow us to model $\mathcal{D}_{\mathcal{A}}$ as a partial function from $Q_{\mathcal{A}} \times acts(\mathcal{A})$ to $Disc(Q_{\mathcal{A}})$. However, we keep this presentation to stay as close as possible to the usual notation of the literature. For the same reasons, we use both $\mathcal{A} \triangleq (Q_{\mathcal{A}}, \bar{q}_{\mathcal{A}}, sig(\mathcal{A}), D_{\mathcal{A}})$ and $\mathcal{A} \triangleq (states(\mathcal{A}), start(\mathcal{A}), sig(\mathcal{A}), dtrans(\mathcal{A}))$

Definition 2.4 (Composition of Signatures). Let $\Sigma = (in, out, int)$ and $\Sigma' = (in', out', int')$ be compatible signatures. Then we define their composition $\Sigma \times \Sigma = (in \cup in' - (out \cup out'), out \cup out', int \cup int')$[2].

Signature composition is clearly commutative and associative. Now we can define the compatibility of several automata at a state with the compatibility of their attached signatures. First we define compatibility at a state, and discrete transition for a set of automata for a particular compatible state.

Definition 2.5 (compatibility at a state). Let $\mathbf{A} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$ be a set of PSIOA. A *state* of $\mathbf{A}$ is an element $q = (q_1, ..., q_n) \in Q_{\mathbf{A}} = Q_{\mathcal{A}_1} \times ... \times Q_{\mathcal{A}_n}$. We say $\mathcal{A}_1, ..., \mathcal{A}_n$ are (resp. $\mathbf{A}$ is) *compatible at state* q if $\{sig(\mathcal{A}_1)(q_1), ..., sig(\mathcal{A}_n)(q_n)\}$ is a set of compatible signatures. In this case we note $sig(\mathbf{A})(q) = sig(\mathcal{A}_1)(q_1) \times ... \times sig(\mathcal{A}_n)(q_n)$ as per definition 2.4 and we note $\eta_{(\mathbf{A},q,a)} \in Disc(Q_{\mathbf{A}})$, s. t. $\forall a \in \widehat{sig}(\mathbf{A})(q), \eta_{(\mathbf{A},q,a)} = \eta_1 \otimes ... \otimes \eta_n$ where $\forall j \in [1, n], \eta_j = \eta_{(\mathcal{A}_j,q_j,a)}$ if $a \in sig(\mathcal{A}_j)(q_j)$ and $\eta_j = \delta_{q_j}$ otherwise.

2.4 Hiding and renaming

We present the classic hiding and renaming operators. The former "hides" the output actions transforming them into internal actions.

Definition 2.6 (hiding on signature). Let $sig = (in, out, int)$ be a signature and S a set of actions. We note $hide(sig, S)$ the signature $sig' = (in, out \setminus S, int \cup (out \cap S))$

Definition 2.7 (hiding on PSIOA). Let $\mathcal{A} = (Q_{\mathcal{A}}, \bar{q}_{\mathcal{A}}, sig(\mathcal{A}), D_{\mathcal{A}})$ be a PSIOA. Let h a function mapping each state $q \in Q$ to a set of output actions. We note $hide(\mathcal{A}, h)$ the PSIOA $(Q, \bar{q}, sig'(\mathcal{A}), D)$, where $sig'(\mathcal{A}) : q \in Q \mapsto hide(sig(\mathcal{A})(q), h(q))$.

The next operation simply renames actions of an automaton.

Definition 2.8 (Action renaming for PSIOA). Let $\mathcal{A}$ be a PSIOA and let r be a partial function on $states(\mathcal{A}) \times acts(\mathcal{A})$, s. t. $\forall q \in states(\mathcal{A})$, $r(q)$ is an injective mapping with $\widehat{sig}(\mathcal{A})(q)$ as domain. Then $r(\mathcal{A})$ is the PSIOA (see appendix A, lemma A.1) given by:

(1) $start(r(\mathcal{A})) = start(\mathcal{A})$.
(2) $states(r(\mathcal{A})) = states(\mathcal{A})$.
(3) $\forall q \in states(\mathcal{A})$, $sig(r(\mathcal{A}))(q) = (in(r(\mathcal{A}))(q), out(r(\mathcal{A}))(q), int(r(\mathcal{A}))(q))$ with (a) $out(r(\mathcal{A}))(q) = r(out(\mathcal{A})(q))$, (b) $in(r(\mathcal{A}))(q) = r(in(\mathcal{A})(q))$, (c) $int(r(\mathcal{A}))(q) = r(int(\mathcal{A})(q))$.
(4) $dtrans(r(\mathcal{A})) = \{(q, r(a), \eta) | (q, a, \eta) \in dtrans(\mathcal{A})\}$ (we note $\eta_{(r(\mathcal{A}),q,r(a))}$ the element of $Disc(states(r(\mathcal{A}))$ which is equal to $\eta_{(\mathcal{A},q,a)}$.

2.5 Probabilistic Configuration Automata

The framework proposed in [7] combines the notion of configuration of [1] with the probabilistic setting of [14]. A configuration is a set of automata attached with their current states.

Definition 2.9 (Configuration). A configuration is a pair $(\mathbf{A}, \mathbf{S})$ where

- $\mathbf{A} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$ is a finite set of PSIOA identifiers and
- $\mathbf{S}$ maps each $\mathcal{A}_k \in \mathbf{A}$ to an $s_k \in states(\mathcal{A}_k)$.

In distributed computing, configuration usually refers to the union of states of **all** the automata of the system. Here, there is a subtlety, since it captures a set of some automata ($\mathbf{A}$) in their current state ($\mathbf{S}$), but the set of automata of the systems will not be fixed in the time.

Definition 2.10 (Compatible configuration). A configuration $(\mathbf{A}, \mathbf{S})$, with $\mathbf{A} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$, is compatible iff the set $\mathbf{A}$ is compatible at state $(\mathbf{S}(\mathcal{A}_1), ..., \mathbf{S}(\mathcal{A}_n))$ as per definition 2.5

Definition 2.11 (Intrinsic attributes of a configuration). Let $C = (\mathbf{A}, \mathbf{S})$ be a compatible configuration. Then we define

- $auts(C) = \mathbf{A}$ represents the automata of the configuration,
- $map(C) = \mathbf{S}$ maps each automaton of the configuration with its current state,
- $sig(C) = (in(C), out(C), int(C)) = ([(\bigcup_{\mathcal{A} \in \mathbf{A}} in(\mathcal{A})(\mathbf{S}(\mathcal{A}))) \setminus out(C)], \bigcup_{\mathcal{A} \in \mathbf{A}} out(\mathcal{A})(\mathbf{S}(\mathcal{A})), \bigcup_{\mathcal{A} \in \mathbf{A}} int(\mathcal{A})(\mathbf{S}(\mathcal{A})))$, is called the intrinsic signature of the configuration,

Here we define a reduced configuration as a configuration deprived of the automata that are in the very particular state where their current signatures are the empty set. This mechanism will be used later to capture the idea of destruction of an automaton.

Definition 2.12 (Reduced configuration). $reduce(C) = (\mathbf{A}', \mathbf{S}')$, where $\mathbf{A}' = \{\mathcal{A} | \mathcal{A} \in \mathbf{A} \text{ and } sig(\mathcal{A})(\mathbf{S}(\mathcal{A})) \neq \emptyset\}$ and $\mathbf{S}'$ is the restriction of $\mathbf{S}$ to $\mathbf{A}'$, noted $\mathbf{S} \upharpoonright \mathbf{A}'$ in the remaining.
A configuration C is a reduced configuration iff $C = reduce(C)$.

We will define some probabilistic transition from configurations to others where some automata can be destroyed or created. To define it properly, we start by defining "preserving transition" where no automaton is neither created nor destroyed and then we define above this definition the notion of configuration transition. These distributions belong to the measurable set $(Q_{conf}, Disc(Q_{conf}))$ where Q_{conf} denotes the (countable) set of configurations.
We start by defining preserving transition (C, a, η_p) from a configuration C via an action a with a transition $\eta_p \in Disc(Q_{conf})$. It allows us to say what is the "static" probabilistic transition from a configuration C via an action a if no creation or destruction occurs.

Definition 2.13 (preserving transition). Let $\mathbf{A} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$ be a finite set of automata Let $C = (\mathbf{A}, \mathbf{S})$ be a compatible configuration s. t. $q = US(C)$ and $a \in \widehat{sig}(C)$. Let $\eta_p \in Disc(Q_{conf})$.
We note $C \xrightarrow{a} \eta_p$ if $\eta_{(\mathbf{A},q,a)}$ and η_p verify the following:

- $\forall (\mathbf{A}', \mathbf{S}') \in supp(\eta_p)$, $\mathbf{A}' = \mathbf{A}$
- $\forall (\mathbf{A}, \mathbf{S}') \in Q_{conf}$, $\forall q' = (q'_1, ..., q'_n) \in states(\mathbf{A})$, then $(\forall i \in [1, n], q'_i = \mathbf{S}'(\mathcal{A}_i)) \implies \eta_{(\mathbf{A},q,a)}(q') = \eta_p((\mathbf{A}, \mathbf{S}'))$.

Now we are ready to define our "dynamic" transition, that allows a configuration to create or destroy some automata.

Definition 2.14 (Intrinsic transition). Let $(\mathbf{A}, \mathbf{S})$ be arbitrary reduced compatible configuration, let $\eta \in Disc(Q_{conf})$, and let $\varphi \subseteq Autids$, $\varphi \cap \mathbf{A} = \emptyset$ and φ is finite. Then $(\mathbf{A}, \mathbf{S}) \xRightarrow{a}_{\varphi} \eta$ if it exists $\eta_p, \eta_{nr} \in Disc(Q_{conf})$ s. t. $(\mathbf{A}, \mathbf{S}) \xrightarrow{a} \eta_p$ and

- (φ is created with probability 1)

[2] not to be confused with Cartesian product. We keep this notation to stay as close as possible to the literature.

$\forall(\mathbf{A}'', \mathbf{S}'') \in supp(\eta_{nr})$, $\mathbf{A}'' = \mathbf{A} \cup \varphi$.[3]

- (freshly created automata start at start state) $\forall C'' = (\mathbf{A}'', \mathbf{S}'') \in supp(\eta_{nr})$, $\forall \mathcal{A}_i \in \varphi \setminus \mathbf{A}$, $\mathbf{S}''(\mathcal{A}_i) = start(\mathcal{A}_i)$
- (The non-reduced transition match the preserving transition) $\forall(\mathbf{A}'', \mathbf{S}'') \in Q_{conf}$, s. t. $\mathbf{A}'' = \mathbf{A} \cup \varphi$ and $\forall \mathcal{A}_i \in \varphi \setminus \mathbf{A}$, $\mathbf{S}''(\mathcal{A}_i) = start(\mathcal{A}_i)$, $\eta_{nr}((\mathbf{A}'', \mathbf{S}'')) = \eta_p(\mathbf{A}, \mathbf{S}''\lceil\mathbf{A}))$ where $\mathbf{S}''\lceil\mathbf{A}$ denotes the restriction of $\mathbf{S}''$ on $\mathbf{A}$)
- (The reduced transition match the non-reduced transition) $\forall c' \in Q_{conf}$, if $c' = reduce(c')$, $\eta_r(c') = \Sigma_{(c'', c'=reduce(c''))} \eta_{nr}(c'')$, if $c' \neq reduce(c')$, then $\eta_r(c') = 0$

Here below we recall the definition of probabilistic configuration automata [7] altogether with notations to represent corresponding probability measures.

Definition 2.15 ($\eta \xleftrightarrow{f} \eta'$). Let Q and Q' be two countable sets. Let $(\eta, \eta') \in Disc(Q) \times Disc(Q')$. Let $f : Q \to Q'$. We note $\eta \xleftrightarrow{f} \eta'$ if the following is verified:

- the restriction $\tilde{f}$ of f to $supp(\eta)$ is a bijection from $supp(\eta)$ to $supp(\eta')$
- $\forall q \in supp(\eta)$, $\eta(q) = \eta'(f(q))$

Now we are ready to define our probabilistic configuration automata. Such an automaton define a strong link with a dynamic configuration.

Definition 2.16 (Probabilistic Configuration Automaton). A probabilistic configuration automaton (PCA) X consists of the following components:

- 1. A probabilistic signature I/O automaton $psioa(X)$. For brevity, we define $states(X) = states(psioa(X))$, $start(X) = start(psioa(X))$, $sig(X) = sig(psioa(X))$ and likewise for all other (sub)components and attributes of $psioa(X)$.
- 2. A configuration mapping $config(X)$ with domain $states(X)$ and such that $config(X)(q_X)$ is a reduced compatible configuration for all $q_X \in states(X)$.
- 3. For each $q_X \in states(X)$, a mapping $created(X)(q_X)$ with domain $sig(X)(q_X)$ and such that $\forall a \in sig(X)(q_X)$, $created(X)(q_X)(a) \subseteq Autids$ with $created(X)(q_X)(a)$ finite.
- 4. A hidden-actions mapping $hidden\text{-}actions(X)$ with domain $states(X)$ and such that $hidden\text{-}actions(X)(q_X) \subseteq out(config(X)(q_X))$.

and satisfies the following constraints

- 1. (start states preservation) If $config(X)(\bar{q}_X) = (\mathbf{A}, \mathbf{S})$, then $\forall \mathcal{A}_i \in \mathbf{A}, \mathbf{S}(\mathcal{A}_i) = \bar{q}_i$
- 2. (top/down simulation) If $(q_X, a, \eta_{(X,q_X,a)}) \in dtrans(X)$ then it exists $\eta' \in Disc(Q_{conf})$ s. t. $\eta_{(X,q_X,a)} \xleftrightarrow{f} \eta'$ with i) $f = config(X)$ and ii) $config(X)(q_X) \overset{a}{\Longrightarrow}_{\varphi} \eta'$, where $\varphi = created(X)(q_X)(a)$
- 3. (bottom/up simulation) If $C \overset{a}{\Longrightarrow}_{\varphi} \eta'$ with $q_X \in states(X)$, $C = config(X)(q_X)$, $a \in \widehat{sig}(C)$, $\varphi = created(X)(x)(a)$ and $\eta' \in Disc(Q_{conf})$, then $(q_X, a, \eta_{(X,q_X,a)}) \in dtrans(X)$, and $\eta_{(X,q_X,a)} \xleftrightarrow{f} \eta'$ with $f = config(X)$.
- 4. (action hiding) for every $q_X \in states(X)$, $sig(X)(q_X) = hide(sig(config(X)(q_X)), hidden\text{-}actions(q_X))$.

This definition, proposed in a deterministic fashion in [1], captures dynamicity of the system. Each state is linked with a configuration. The set of automata of the configuration can change during an execution. A sub-automaton $\mathcal{A}$ is created from state q by the action a if $\mathcal{A} \in created(X)(q)(a)$. A sub-automaton $\mathcal{A}$ is destroyed if the non-reduced attached configuration distribution lead to a configuration where $\mathcal{A}$ is in a state $q^{\phi}_{\mathcal{A}}$ s. t. $\widehat{sig}(\mathcal{A})(q^{\phi}_{\mathcal{A}}) = \emptyset$. Then the corresponding reduced configuration will not hold $\mathcal{A}$. The last constraint states that the signature of a state q_X of X must be the same as the signature of its corresponding configuration $config(X)(q_X)$, except for the possible effects of hiding operators, so that some outputs of $config(X)(q_X)$ may be internal actions of X in state q_X.

Definition 2.17 (hiding on PCA). Let X be a PCA. Let $h : q \in states(X) \mapsto h(q) \subset out(X)(q)$ a function mapping each state $q \in states(X)$ to a set of output actions. We note $hide(X, h)$ the PCA X' that differs from X only on $sig(X')$ and $hidden\text{-}actions(X')$, where $\forall q \in states(X) = states(X')$,

- $sig(X')(q) = hide(sig(X)(q), h(q))$ and
- $hidden\text{-}actions(X')(q) = hidden\text{-}actions(X)(q) \cup h(q)$.

2.6 PSIOA and PCA composition

It is possible to extend definition of executions to a set $\mathbf{A}$ of PSIOA (resp. PCA) in the obvious way, i. e. with alternating sequence of compatible states (except potentially for last state) and actions of $\mathbf{A}$. Thus we can define reachable states of $\mathbf{A}$ in a natural manner, before defining partially-compatible set of PSIOA (resp. PCA) as a set PSIOA (resp. PCA) s. t. every reachable state is compatible. More details can be found in [7]. This precautions allow us to formally define our operation of composition. This is the central operation of any IOA formalism.

Definition 2.18 (PSIOA partial-composition). If $\mathbf{A} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$ with $\mathcal{A}_i = (Q_{\mathcal{A}_i}, \bar{q}_{\mathcal{A}_i}, sig(\mathcal{A}_i), D_{\mathcal{A}_i})$, is a partially-compatible set of PSIOA, then their partial-composition $\mathcal{A}_1||...||\mathcal{A}_n$, is defined to be $\mathcal{A} = (Q_{\mathcal{A}}, \bar{q}_{\mathcal{A}}, sig(\mathcal{A}), D_{\mathcal{A}})$, where:

- $Q_{\mathcal{A}} = \{q \in Q_{\mathcal{A}_1} \times ... \times Q_{\mathcal{A}_n} | q \text{ is a reachable state of } \mathbf{A}\}$.
- $\bar{q}_{\mathcal{A}} = (\bar{q}_{\mathcal{A}_1}, ..., \bar{q}_{\mathcal{A}_n})$
- $sig(\mathcal{A}) : q = (q_1, ..., q_n) \in Q_{\mathcal{A}} \mapsto sig(\mathcal{A})(q) = sig(\mathcal{A}_1)(q_1) \times ... \times sig(\mathcal{A}_n)(q_n)$ as per definition 2.4.
- $D_{\mathcal{A}} \subset Q_{\mathcal{A}} \times acts(\mathcal{A}) \times Disc(Q_{\mathcal{A}})$ is the set of triples of the form $(q, a, \eta_{(\mathbf{A},q,a)})$ with $q \in Q_{\mathcal{A}}$ and $a \in \widehat{sig}(\mathbf{A})(q)$

If $q = (q_1, ..., q_n) \in states(\mathcal{A})$, we note $q \upharpoonright \mathcal{A}_i \triangleq q_i$ the projection on the i-th element of q.

Definition 2.19 (PCA partial-composition). If $\mathbf{X} = \{X_1, ..., X_n\}$ with $cr_i = created(X_i)$, is a partially-compatible set of PCA, then their partial-composition $X_1||...||X_n$, is defined to be the PCA X s. t. $psioa(X) = psioa(X_1)||...||psioa(X_n)$ and $\forall q \in states(X)$:

- $config(X)(q) = \bigcup_{i \in [1,n]} config(X_i)(q \upharpoonright X_i)$

[3] The assumption of deterministic creation is not restrictive, nothing prevents from flipping a coin at state s_0 to reach s_1 with probability p or s_2 with probability $1 - p$ and only create a new automaton in state s_2 with probability 1, while the action create is not enabled in state s_1.

- $created(X)(q)(a) = \bigcup_{i\in[1,n]} created(X_i)(q \upharpoonright X_i)(a), \forall a \in \widehat{sig}(X)(q)$, with the convention $created(X_i)\ (q_i)\ (a) = \emptyset$ if $a \notin \widehat{sig}(X_i)(q_i)$
- $hidden\text{-}actions(q) = \bigcup_{i\in[1,n]} hidden\text{-}actions(X_i)(q \upharpoonright X_i)$

The set of PSIOA (resp. PCA) has been shown to be close under partial composition in [7].

3 SCHEDULER AND EXTERNAL PERCEPTION

In this section we recall the definitions of scheduler and environment of the [7] framework.

An inherent non-determinism appears for composable (I/O) automata. Indeed, after composition (or even before), it is natural to obtain a state with several enabled actions. The most common case is the reception of two concurrent messages in flight from two different processes. This non-determinism must be solved if we want to define a probability measure on the automata executions and be able to say that a situation is likely to occur or not. To solve the non-determinism, we use a scheduler that chooses an enabled action from a signature.

A scheduler is hence a function that takes an execution fragment as input and outputs the probability distribution on the set of transitions that will be triggered. In the following we recall definition in [7].

Definition 3.1 (scheduler). A *scheduler* of a PSIOA $\mathcal{A}$ is a function $\sigma : Frags^*(\mathcal{A}) \rightarrow SubDisc(dtrans(\mathcal{A}))$ such that $(q, a, \eta) \in supp(\sigma(\alpha))$ implies $q = lstate(\alpha)$. Here $SubDisc(dtrans(\mathcal{A}))$ is the set of discrete sub-probability distributions on $dtrans(\mathcal{A})$. Loosely speaking, σ decides (probabilistically) which transition to take after each finite execution fragment α. Since this decision is a discrete sub-probability measure, it may be the case that σ chooses to halt after α with non-zero probability: $1 - \sigma(\alpha)(dtrans(\alpha)) > 0$. We note $schedulers(\mathcal{A})$ the set of schedulers of $\mathcal{A}$.

It is possible to use a scheduler σ to generate a measure of probability ϵ_σ on the measurable space $(Execs(\mathcal{A}), \mathcal{F}_{Execs(\mathcal{A})})$ where $\mathcal{F}_{Execs(\mathcal{A})}$ is the sigma-field generated by cones of execution fragments, where each cone $C_{\alpha'}$ is the set of execution fragments that have α' as a prefix, i. e. $C_{\alpha'} = \{\alpha \in Execs(\mathcal{A}) | \alpha' \leq \alpha\}$. More details are available in [3] and [7].

Without restriction, a scheduler could become a too powerful adversary for practical applications. Hence, it is common to only consider a subset of schedulers, called a *scheduler schema*, like "oblivious schedulers", "off-line schedulers", "fair schedulers", "task-schedulers", ...

Definition 3.2 (scheduler schema). A *scheduler schema* is a function that maps any PSIOA or PCA W to a subset of $schedulers(W)$.

In the following we recall the notion of *implementation* [7]. The intuition behind this notion is the fact that any environment $\mathcal{E}$ that would interact with both $\mathcal{A}$ and $\mathcal{B}$, would not be able to distinguish $\mathcal{A}$ from $\mathcal{B}$. The classic use-case is to formally show that a (potentially very sophisticated) algorithm implements a specification.

Definition 3.3 (Environment). A probabilistic environment for PSIOA $\mathcal{A}$ is a PSIOA $\mathcal{E}$ such that $\mathcal{A}$ and $\mathcal{E}$ are partially-compatible. We note $env(\mathcal{A})$ the set of environments of $\mathcal{A}$.

Now we define *insight function* which is a function that captures the insights that could be obtained by an external observer to attempt a distinction.

Definition 3.4 (insight function). An *insight-function* is a function $f_{(.,.)}$ parametrized by a pair $(\mathcal{E}, \mathcal{A})$ of PSIOA where $\mathcal{E} \in env(\mathcal{A})$ so that for every PSIOA $\mathcal{E}$, it exists a measurable space $(G_\mathcal{E}, \mathcal{F}_{G_\mathcal{E}})$, s. t. for every PSIOA $\mathcal{A}$ where $\mathcal{E} \in env(\mathcal{A})$, $f_{(\mathcal{E},\mathcal{A})}$ is a measurable function from $(Execs(\mathcal{E}||\mathcal{A}), \mathcal{F}_{Execs(\mathcal{E}||\mathcal{A})})$ to $(G_\mathcal{E}, \mathcal{F}_{G_\mathcal{E}})$.

The point is that the arrival space $(G_\mathcal{E}, \mathcal{F}_{G_\mathcal{E}})$ is the same for two different functions $f_{(\mathcal{E},\mathcal{A})}$ and $f_{(\mathcal{E},\mathcal{B})}$ to enable a comparison. Some examples of insight-functions are the trace function, the print function introduced in [7] and the function *accept* introduced in [3].

Since an insight-function $f_{(.,.)}$ is measurable, we can define the image measure of ϵ_σ under $f_{(\mathcal{E},\mathcal{A})}$, i. e. the probability to obtain a certain external perception under a certain scheduler σ.

Definition 3.5 (f-dist). Let $f_{(.,.)}$ be an insight-function. Let $(\mathcal{E}, \mathcal{A})$ be a pair of PSIOA where $\mathcal{E} \in env(\mathcal{A})$. Let $\sigma \in schedulers(\mathcal{E}||\mathcal{A})$. We define $f\text{-}dist_{(\mathcal{E},\mathcal{A})}(\sigma)$, to be the image measure of ϵ_σ under $f_{(\mathcal{E},\mathcal{A})}$ (the function that maps any $C \in \mathcal{F}_{G_\mathcal{E}}$ to $\epsilon_\sigma(f^{-1}_{(\mathcal{E},\mathcal{A})}(C))$) .

Now we define the notion of *balanced schedulers*, that capture the incapacity of an environment to distinguish two situations under two so-called balanced schedulers. Several definitions of this incapacity can be chosen.

Definition 3.6 (balanced schedulers). Let $\mathcal{A}$ and $\mathcal{B}$ be two PSIOA (resp. PCA), let $\mathcal{E} \in env(\mathcal{A}) \cap env(\mathcal{B})$, let $(\sigma, \sigma') \in schedulers(\mathcal{E}||\mathcal{A}) \times schedulers(\mathcal{E}||\mathcal{B})$. Let $\epsilon \in \mathbb{R}^{\geq 0}$. Let $f_{(,)}$ be an insight function. We note $\sigma S^{\leq\epsilon}_{\mathcal{E},f}\sigma'$ if $\forall I \subseteq N, \forall(\zeta_i)_{i\in I} \in [\bigcup_{C\in autids,\mathcal{E}\in env(C)} range(f_{(\mathcal{E},C)})]^I, |\sum_{i\in I}(f\text{-}dist_{(\mathcal{E},\mathcal{B})}(\sigma')(\zeta_i) - f\text{-}dist_{(\mathcal{E},\mathcal{A})}(\sigma)(\zeta_i))| \leq \epsilon$.

We states a necessary and sufficient condition to obtain composability of f-implementation stated in lemma 4.13and 4.14. This condition of stability by composition is an extension to approximate implementation of the one introduced in [7].

Definition 3.7 (insight function stable by composition). Let $f_{(.,.)}$ be an insight-function. We say that $f_{(.,.)}$ is *stable by composition* if for every quadruplet of PSIOA $(\mathcal{A}_1, \mathcal{A}_2, \mathcal{B}, \mathcal{E})$, s. t. $\mathcal{B}$ is partially compatible with $\mathcal{A}_1$ and $\mathcal{A}_2$, $\mathcal{E} \in env(\mathcal{B}||\mathcal{A}_1) \cap env(\mathcal{B}||\mathcal{A}_2)$, $\forall \epsilon \in \mathbb{R}^{\geq 0}$, $\forall(\sigma, \sigma') \in schedulers(\mathcal{E}||\mathcal{B}||\mathcal{A}_1) \times schedulers(\mathcal{E}||\mathcal{B}||\mathcal{A}_2)$, $\sigma S^{\leq\epsilon}_{\mathcal{E}||\mathcal{B},f}\sigma'$ implies $\sigma S^{\leq\epsilon}_{\mathcal{E},f}\sigma'$.

This very modest property is naturally verified by standard insight functions like *trace*, *accept* [3], or *print* [7] and captures the fact that the environment $\mathcal{E}$ has not a greater power of distinction than $\mathcal{E}||\mathcal{B}$, which should be verified by any reasonable insight function.

4 POLYNOMIAL-BOUNDED PSIOA: FORMALISE COMPUTATIONAL INDISTINGUISHABILITY

In previous sections, we recall the key components of probabilistic dynamic input/output automata introduced in [7] that provides a semantic for probabilistic concurrent computation and allows systems to dynamically create new protocol instances at run time

(e.g. dynamic ITM [10] invocation mechanism in the UC framework [2] or through the bang operator "!" in the IITM [11] framework). In this section, we extend the approximate implementation relation defined in [4] to dynamic settings, to express the idea that every behavior of one family of dynamic automata is computationally indistinguishable from some behavior of another dynamic automata family.

We adopt a standard bit-representation where we note $\langle q \rangle$, $\langle a \rangle$, $\langle tr \rangle$, $\langle C \rangle$ the respective bit-string representations of state q, action a, discrete transition tr and configuration C.

4.1 b-bounded PSIOA

In the following we extend the definition of bounded Task-PIOA [4] to dynamic settings. We define bounded PSIOA and then bounded PCA. The idea is to both limit the memory and the computational power of the concerned PSIOA. Typically, we prohibit transitions that would implicitly violate some computational hardness assumptions.

Definition 4.1 (PSIOA b-time-bounded). PSIOA $\mathcal{A}$ is said to be b-time-bounded, where $b \in \mathbb{R}^{\geq 0}$, provided that:

(1) Automaton parts: The length of the bit-string representation of every action, state, transition is at most b.
(2) Decoding: There exist deterministic Turing machine M_{start}, M_{sig}, M_{trans}, M_{step} so that, (i) given the representation $\langle q \rangle$ of a candidate state q, M_{start} decides whether q is the unique start state of $\mathcal{A}$, (ii) given the representation $\langle q \rangle$ of a state $q \in states(\mathcal{A})$, given the representation $\langle a \rangle$ of a candidate input action, (resp. output action, resp. internal action) M_{sig} decides whether $a \in in(\mathcal{A})(q)$ (resp. $a \in out(\mathcal{A})(q)$, resp. $a \in int(\mathcal{A})(q)$)[4], (iii) given the representation $\langle q \rangle$ of a state $q \in states(\mathcal{A})$, the representation $\langle a \rangle$ of an action $a \in \widehat{sig}(\mathcal{A})(q)$ and the representation $\langle tr \rangle$ of $tr = (q, a, \eta)$, M_{trans} decides whether $tr \in D_{\mathcal{A}}$ and (iv) given the representation $\langle tr \rangle$ of $tr = (q, a, \eta) \in D_{\mathcal{A}}$ and the representation $\langle q' \rangle$ of a candidate states, M_{step} decides whether $(q, a, q') \in steps(\mathcal{A})$ (i. e. $q' \in supp(\eta)$). Moreover all this machines run always in time at most b.
(3) Determining the next state: There is a probabilistic Turing machine $Mstate$ that, given the representation $\langle q \rangle$ of a state q of $\mathcal{A}$, and the representation $\langle a \rangle$ of an action $a \in \widehat{sig}(\mathcal{A})(q)$ that is enabled in q, produces the representation $\langle q' \rangle$ of the next state q' resulting from the unique transition of $\mathcal{A}$ of the form (q, a, η). Moreover, Mstate always runs in time at most b. Moreover, we require that every Turing machine mentioned in this definition can be described using a bit string of length at most b, according to some standard encoding of Turing machines.

We naturally extend the last definition 4.1 to PCA.

Definition 4.2 (PCA b-time-bounded). PCA X is said to be b-time-bounded, where $b \in \mathbb{R}^{\geq 0}$, provided that:

- $psioa(X)$ is b-time-bounded as per definition 4.1
- for every $q \in states(X)$, for every $a \in \widehat{sig}(X)(q)$ the length of bit-string representation of $config(X)(q)$, $hidden\text{-}actions(X)(q)$, $created(X)(q)(a)$ are at most b.
- There is a deterministic Turing machine M_{conf} (resp. $M_{created}$, resp. M_{hidden}) that, given the representation $\langle q \rangle$ of $q \in states(X)$, the representation $\langle a \rangle$ of $a \in \widehat{sig}(X)(q)(a)$, outputs the representation $\langle C \rangle$ (resp. $\langle \varphi \rangle$, resp. $\langle h \rangle$) of $C = config(X)(q)$ (resp. $\varphi = created(X)(q)(a)$, resp. $h = hidden\text{-}actions(X)(q)$). The 3 machines run always in time b and can be described with a bit-string of length at most b using standard encoding of Turing machines.

4.2 Composition

As for bounded task-PIOA [4], the composition of two time-bounded PSIOA (resp. PCA) is also time-bounded, with a bound that is a linear combination of the bounds for the two components.

Lemma 4.3 (composition of bounded PSIOA is bounded). *There exists a constant c_{comp} such that the following holds. Suppose $\mathcal{A}_1$ is a b_1-time-bounded PSIOA (resp. PCA) and $\mathcal{A}_2$ is a b_2-time-bounded PSIOA (resp. PCA), where $b_1, b_2 \geq 1$. Then $\mathcal{A}_1 || \mathcal{A}_2$ is a $c_{comp} \cdot (b_1 + b_2)$-bounded PSIOA (resp. PCA).*

Proof. The proof is delegated to the appendix B (see lemma B.1 and B.2). □

4.3 Hiding

In order to be able to state time bounds when we apply the hiding operator on a time-bounded PSIOA (resp. PCA) $\mathcal{A}$, we define what it means for a set of actions of $\mathcal{A}$ to be b-time recognizable.

Definition 4.4 (bounded-time recognizable set/function). Suppose S' is a set of actions and $b' \in \mathbb{R}^{\geq 0}$. We say that S is *b'-time recognizable* if there is a probabilistic Turing machine M' that, given the representation $\langle a \rangle$ of a candidate action a, decides if $a \in S$. Furthermore, the machine M' runs in time less than b' and can be described by less than b' bits, according to some standard encoding.

Let $\mathcal{A}$ be a bounded-PSIOA (resp. PCA). Suppose S is a function mapping each state $q \in states(\mathcal{A})$ to a set of states $S(q)$. We say that S is b'-time recognizable if $\forall q \in states(\mathcal{A})$, $S(q)$ is b'-time recognizable.

Lemma 4.5 (hiding of bounded automata is bounded). *There exists a constant c_{hide} such that the following holds. Suppose $\mathcal{A}$ is a b-time-bounded PSIOA (resp. PCA), where $b \in \mathbb{R}^{\geq 0}$, $b \geq 1$. Let S be a b'-time recognizable function with $states(\mathcal{A})$ as domain. Then $hide(\mathcal{A}, S)$ is a $c_{hide} \cdot (b + b')$-time-bounded PSIOA (resp. PCA).*

Proof. The proof is delegated to the appendix B (see B.3). □

4.4 Scheduling

In previous section, we adapted the boundness of I/O automata from [4] into dynamic setting. However, for the moment, there is no a priori bound imposed on the number of transitions that a PSIOA or a PCA may perform, which could lead to a potential unbounded behaviour and so to a potentially too important computational power.

Therefore [4] introduced a final restriction on runtime imposed only for comparison of the behaviors of different PSIOA (resp. PCA)

[4] if $a \in \widehat{sig}(\mathcal{A})(q)$, a is enabled by assumption $\mathbf{E}_1$ of action enabling

using implementation relations, by adding bounds on the number of activations.

In this paper, we are slightly less restrictive than [4], since we tolerate a broader set of schedulers instead of only accepting task-schedulers [3] which generalizes fully off-line schedulers that decide in advance an order of "tasks" to perform, where a task is an equivalence class on actions. In addition of the obtained generality, the advantages are as follows:

- We do not have to formalise the extension of task-structures to dynamic setting with the attached issues mentioned in [7].
- We can define a scheduler schema that is *oblivious* in the sufficient sense to ensure the correctness of the studied emulation candidate.
- We can define a *creation-oblivious* scheduler schema. This property has been shown in [7] to be necessary to ensure that the implementation relation is monotonic w.r.t. PSIOA creation, i. e. if PCA $X_{\mathcal{A}}$ and $X_{\mathcal{B}}$ differ only in that $X_{\mathcal{A}}$ dynamically creates and destroys PSIOA $\mathcal{A}$ instead of creating and destroying PSIOA $\mathcal{B}$ as $X_{\mathcal{B}}$ does, and if $\mathcal{A}$ implements $\mathcal{B}$ (in the sense they cannot be distinguished by any external observer), then $X_{\mathcal{A}}$ implements $X_{\mathcal{B}}$. This property will allow us to obtain monotonicity w.r.t. PSIOA creation for the relation of secure-emulation in a future work.

Definition 4.6 (bounded scheduler). Let K be a PSIOA or a PCA, let $b \in \mathbb{N}$. Let $\sigma \in schedulers(K)$, we say that σ is b-time bounded if $\forall \alpha \in execs(K)$ with $|\alpha| > b$, $supp(\sigma(\alpha)) = \emptyset$, i. e. the scheduler never execute more than b actions of K.

We could require that (*) the scheduler has to be a bounded automaton, but this precision is at the discretion of the designers of the solution. The results remains true if (*) is required or not.

4.5 Polynomially-bounded family of PSIOA

In this section we extend bounds to families of PSIOA and PCA.

Definition 4.7 (PSIOA family). A PSIOA (resp. PCA) family $\underline{\mathcal{A}}$, is an indexed set, $(\mathcal{A}_k)_{k\in\mathbb{N}}$, of PSIOA (resp. PCA). Two PSIOA (resp. PCA) families $\underline{\mathcal{A}} = (\mathcal{A}_k)_{k\in\mathbb{N}}$ and $\underline{\mathcal{B}} = (\mathcal{B}_k)_{k\in\mathbb{N}}$ are said to be compatible provided that, for every k, $\mathcal{A}_k$ and $\mathcal{B}_k$ are compatible. Two PSIOA (resp. PCA) families $\underline{\mathcal{A}} = (\mathcal{A}_k)_{k\in\mathbb{N}}$ and $\underline{\mathcal{B}} = (\mathcal{B}_k)_{k\in\mathbb{N}}$ can be composed to yield $\underline{C} = (C_k)_{k\in\mathbb{N}} = \underline{\mathcal{A}}||\underline{\mathcal{B}}$ by defining $C_k = \mathcal{A}_k||\mathcal{B}_k$ for every $k \in \mathbb{N}$.

Definition 4.8 (Time-Bounded PSIOA Family). The PSIOA (resp. PCA) family $\underline{\mathcal{A}} = (\mathcal{A}_k)_{k\in\mathbb{N}}$ is said to be b-time-bounded, where $b : \mathbb{N} \to \mathbb{R}^{\geq 0}$, provided that $\mathcal{A}_k$ is $b(k)$-time bounded for every $k \in \mathbb{N}$. This definition allows different Turing machines to be used for each $k \in \mathbb{N}$.

Definition 4.9 (scheduler family). A scheduler family $\underline{\sigma}$ is an indexed set, $(\sigma_k)_{k\in\mathbb{N}}$ of schedulers.

Definition 4.10 (bounded scheduler family). Let $\underline{\sigma} = (\sigma_k)_{k\in\mathbb{N}}$ be a scheduler family. Then σ is said to be b-time-bounded, where $b : \mathbb{N} \to \mathbb{R}^{\geq 0}$ provided that σ_k is $b(k)$-time bounded for every k.

4.6 Implementation

In this section, we adapt the approximate implementation to the bounded setting following the same methodology as in [4].

First, we extend the notion of *balanced schedulers* to schedulers family.

Definition 4.11 (balanced schedulers). Let $\underline{\mathcal{A}} = (\mathcal{A}_k)_{k\in\mathbb{N}}$ and $\underline{\mathcal{B}} = (\mathcal{B}_k)_{k\in\mathbb{N}}$ be two PSIOA families or two PCA families, let $\underline{\mathcal{E}} = (\mathcal{E}_k)_{k\in\mathbb{N}} \in env(\underline{\mathcal{A}}) \cap env(\underline{\mathcal{B}})$ (i. e. $\forall k \in \mathbb{N}, \mathcal{E}_k \in env(\mathcal{A}_k) \cap env(\mathcal{B}_k)$), let $(\underline{\sigma} = (\sigma_k)_{k\in\mathbb{N}}, \underline{\sigma'} = (\sigma'_k)_{k\in\mathbb{N}}) \in schedulers(\underline{\mathcal{E}}||\underline{\mathcal{A}}) \times schedulers(\underline{\mathcal{E}}||\underline{\mathcal{B}})$ (i. e. $\forall k \in \mathbb{N}, (\sigma_k, \sigma'_k) \in schedulers(\mathcal{E}_k||\mathcal{A}_k) \times schedulers(\mathcal{E}_k||\mathcal{B}_k)$). Let $\underline{\epsilon} = (\epsilon_k)_{k\in\mathbb{N}} \in (\mathbb{R}^{\leq 0})^{\mathbb{N}}$. Let $f_{(.,.)}$ be an insight function. We note $\underline{\sigma} S^{\leq \underline{\epsilon}}_{\underline{\mathcal{E}},f} \underline{\sigma'}$ if $\forall k \in \mathbb{N}$, $\sigma_k S^{\leq \epsilon_k}_{\mathcal{E}_k,f} \sigma'_k$.

Strongly inspired by classic literature of cryptography, the choice of $f_{(.,.)}$ in [4] is the function *accept* that, given an execution α, outputs 1 if a special action *acc* appears in $trace(\alpha)$ and 0 otherwise. This function captures the idea that the environment distinguishes the real world from the idealized one. This is at the discretion of the user of the framework to chose a particular insight function. But we let it as general as possible to be also able to use the insight function $print_{(.,.)}$ that is particularly well-suited to obtain the monotonicity w.r.t PSIOA creation of implementation [7]. In a future work we want to use the the insight function $print_{(.,.)}$ to extend this monotonicity result to secure emulation.

Now we are ready to extend approximate implementation of [4] to both bounded and dynamic setting.

Definition 4.12 (approximate implementation). Let $\mathcal{A}$ and $\mathcal{B}$ be two PSIOA (resp. PCA). Let Sch be a scheduler schema, $\epsilon \in \mathbb{R}^{\leq 0}$, $(p, q_1, q_2) \in \mathbb{N}^3$ and $f_{(.,.)}$ be an insight function.

We note $\mathcal{A} \leq^{Sch,f}_{p,q_1,q_2,\epsilon} \mathcal{B}$ if for every p-bounded $\mathcal{E} \in env(\mathcal{A}) \cap env(\mathcal{B})$, for every q_1-bounded $\sigma \in Sch(\mathcal{E}||\mathcal{A})$, it exists a q_2-bounded $\sigma' \in Sch(\mathcal{E}||\mathcal{B})$ s. t. $\sigma S^{\leq\epsilon}_{\mathcal{E},f} \sigma'$. We extend this definition to scheduler families as follows.

Let $\underline{\mathcal{A}} = (\mathcal{A}_k)_{k\in\mathbb{N}}$ and $\underline{\mathcal{B}} = (\mathcal{B}_k)_{k\in\mathbb{N}}$ be two PSIOA families or two PCA families, let Sch be a scheduler schema and $(p, q_1, q_2, \epsilon) \in (\mathbb{N} \to \mathbb{N})^3 \times (\mathbb{N} \to \mathbb{R}^{\leq 0})$.

We note $\underline{\mathcal{A}} \leq^{Sch,f}_{p,q_1,q_2,\epsilon} \underline{\mathcal{B}}$ if $\forall k \in \mathbb{N}$, $\mathcal{A}_k \leq^{Sch,f}_{p(k),q_1(k),q_2(k),\epsilon(k)} \mathcal{B}_k$

Finally, we note $\underline{\mathcal{A}} \leq^{Sch,f}_{neg,pt} \underline{\mathcal{B}}$ if it exists $(p, q_1, q_2, \epsilon) \in (\mathbb{N} \to \mathbb{N})^3 \times (\mathbb{N} \to \mathbb{R}^{\leq 0})$ s. t. $\underline{\mathcal{A}} \leq^{Sch,f}_{p,q_1,q_2,\epsilon} \underline{\mathcal{B}}$ where p, q_1, q_2 are polynomial functions and ϵ is a negligible function.

The properties of the underlying dynamic probabilistic I/O layer allow to obtain the composability of the approximate implementation.

Lemma 4.13 (composability $\leq^{Sch,f}_{p,q_1,q_2,\epsilon}$). *Let $\epsilon \in \mathbb{R}^{\geq 0}$ and $p, p_3, q_1, q_2 \in \mathbb{N}$ be given. Let f be an insight function* stable by composition *Let Sch be a a scheduler schema. Let $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ be 3 PSIOA (resp. PCA) satisfying: $\mathcal{A}_3$ has p_3-bounded description and is partially compatible with both $\mathcal{A}_1$ and $\mathcal{A}_2$. Then the following holds.*

If $\mathcal{A}_1 \leq^{Sch,f}_{c_{comp}(p+p_3),q_1,q_2,\epsilon} \mathcal{A}_2$, where c_{comp} is the constant factor associated with description bounds in parallel composition (see lemma 4.3), then $\mathcal{A}_3||\mathcal{A}_1 \leq^{Sch,f}_{p,q_1,q_2,\epsilon} \mathcal{A}_3||\mathcal{A}_2$.

Proof. Fix $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ and all the constants as in the hypotheses. Consider any p-time-bounded environment $\mathcal{E}$ for $\mathcal{A}_3||\mathcal{A}_1$ and $\mathcal{A}_3||\mathcal{A}_2$. We must show that, for every q_1-time-bounded scheduler $\sigma_1 \in Sch(\mathcal{E}||\mathcal{A}_3||\mathcal{A}_1)$, there is a q_2-time-bounded scheduler $\sigma_2 \in Sch(\mathcal{E}||\mathcal{A}_3||\mathcal{A}_2)$ such that $\sigma S^{\leq\epsilon}\sigma'$ To show this, fix σ_1 to be any q_1-time-bounded scheduler in $Sch(\mathcal{E}||\mathcal{A}_3||\mathcal{A}_1)$. The composition $\mathcal{E}||\mathcal{A}_3$ is an environment for both $\mathcal{A}_1$ and $\mathcal{A}_2$. Moreover, lemma 4.3 implies that $\mathcal{E}||\mathcal{A}_3$ is $c_{comp}\cdot(p+p_3)$-time-bounded. Since $\mathcal{A}_1 \leq^{Sch,f}_{c_{comp}\cdot(b+b_3),b_1,b_2} \mathcal{A}_2$, $\mathcal{E}||\mathcal{A}_3$ is a $c_{comp}(b+b_3)$-time-bounded environment for $\mathcal{A}_1$ and $\mathcal{A}_2$, and σ_1 is a q_1-time-bounded scheduler for $\mathcal{E}||\mathcal{A}_3||\mathcal{A}_1$, we know that there is a q_2-time-bounded scheduler σ_2 for $\mathcal{E}||\mathcal{A}_3||\mathcal{A}_2$ such $\sigma_1 S^{\leq\epsilon}_{(\mathcal{E}||\mathcal{A}_3),f}\sigma_2$. Finally, the stability by composition of f gives $\sigma_1 S^{\leq\epsilon}_{(\mathcal{E},f)}\sigma_2$ which ends the proof. □

Let us note that both *accept* and $print_{(,)}$ are insight functions stable by composition. In the sequel we extend the previous result to the implementation of automata family.

Lemma 4.14 (composability $\leq^{Sch,f}_{p,q_1,q_2,\epsilon}$). *Let $\epsilon : \mathbb{N} \to \mathbb{R}^{\geq 0}$ and $p, p_3, q_1, q_2 : \mathbb{N} \to \mathbb{N}$ be given. Let Sch be a scheduler schema. Let $f_{(,,)}$ be an insight function stable by composition. Let $\mathcal{A}$, $\mathcal{B}$ and C be 3 PSIOA (resp. PCA) families satisfying: C has p_3-bounded description and is partially compatible with both $\mathcal{A}$ and $\mathcal{B}$. Let c_{comp} be the constant factor associated with description bounds in parallel composition (see lemma 4.3) Then the following holds.*

If $\mathcal{A} \leq^{Sch,f}_{c_{comp}(p+p_3),q_1,q_2,\epsilon} \mathcal{B}$, then $C||\mathcal{A} \leq^{Sch,f}_{p,q_1,q_2,\epsilon} C||\mathcal{B}$.

Proof. The adaptation of lemma 4.13 to family is straightforward (see appendix B lemma B.5) □

Theorem 4.15 (composability $\leq^{Sch,f}_{neg,pt}$). *Let Sch be a scheduler schema. Let $f_{(,,)}$ be an insight function stable by composition.*

Let $\underline{\mathcal{A}}_1$, $\underline{\mathcal{A}}_2$ and $\underline{\mathcal{A}}_3$ be PSIOAs (resp. PCA) families satisfying: $\underline{\mathcal{A}}_3$ has p_3-bounded description where p_3 is a polynomial and is partially compatible with both $\underline{\mathcal{A}}_1$ and $\underline{\mathcal{A}}_2$. Then the following holds.

If $\underline{\mathcal{A}}_1 \leq^{Sch,f}_{neg,pt} \underline{\mathcal{A}}_2$, then $\underline{\mathcal{A}}_3||\underline{\mathcal{A}}_1 \leq^{Sch,f}_{neg,pt} \underline{\mathcal{A}}_3||\underline{\mathcal{A}}_2$. Observe that, by induction, Theorem generalizes to any constant number of substitutions.

Proof. The adaptation of lemma 4.13 to polynomially-bounded family is straightforward (see appendix B , theorem B.6) □

The implementation relationship is also transitive.

Theorem 4.16 (Implementation transitivity). *Let Sch be a scheduler schema. Let $\epsilon_{12}, \epsilon_{23}\epsilon_{13} \in \mathbb{R}^{\leq 0}$, $p, q_1, q_2, q_3 \in \mathbb{N}$ with $\epsilon_{13} = \epsilon_{12}+\epsilon_{23}$, Let $f_{(,,)}$ be an insight-function. Let $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ be PSIOA, s.t. $\mathcal{A}_1 \leq^{Sch,f}_{p,q_1,q_2,\epsilon_{12}} \mathcal{A}_2$ and $\mathcal{A}_2 \leq^{Sch,f}_{p,q_2,q_3,\epsilon_{23}} \mathcal{A}_3$, then $\mathcal{A}_1 \leq^{Sch,f}_{p,q_1,q_3,\epsilon_{13}} \mathcal{A}_3$.*

Proof. See appendix B, theorem B.4 □

4.7 Structured dynamic I/O Automata

In this section we adapt the security layer of [4], on top of the foundational layer introduced in previous sections. This layer follows the general outline of simulation-based security. In this approach, computational security is captured within the model itself.

First, we extend the PSIOA definition with an additional attribute called the environment action mapping which is a function $EAct_{\mathcal{A}}$ with domain $states(\mathcal{A})$ for every PSIOA $\mathcal{A}$ such that $\forall q \in states(\mathcal{A})$, $EAct_{\mathcal{A}}(q) \subseteq \widehat{ext}(\mathcal{A})(q)$, that captures the idea that some actions are intended to be accessible by the environment while others by the adversary.

Definition 4.17 (Structured PSIOA). A structured PSIOA $\mathcal{A} = ((Q_{\mathcal{A}}, \bar{q}_{\mathcal{A}}, sig(\mathcal{A}), D_{\mathcal{A}}), EAct_{\mathcal{A}})$ where $(Q_{\mathcal{A}}, \bar{q}_{\mathcal{A}}, sig(\mathcal{A}), D_{\mathcal{A}})$ is a PSIOA and $EAct_{\mathcal{A}}$ is a mapping function with domain $Q_{\mathcal{A}}$ such that $\forall q \in Q_{\mathcal{A}}, EAct_{\mathcal{A}}(q) \subseteq \widehat{ext}(\mathcal{A})(q)$.

The adversary action mapping of $\mathcal{A}$ is $AAct_{\mathcal{A}} : q \in states(\mathcal{A}) \mapsto \widehat{ext}(\mathcal{A})(q) \setminus EAct_{\mathcal{A}}(q)$.

For convenience, we also define: (i) $EI_{\mathcal{A}} : q \in states(\mathcal{A}) \mapsto EAct_{\mathcal{A}}(q) \cap in(\mathcal{A})(q)$ (environment inputs), (ii) $EO_{\mathcal{A}} : q \in states(\mathcal{A}) \mapsto EAct_{\mathcal{A}}(q) \cap out(\mathcal{A})(q)$ (environment outputs), (iii) $AI_{\mathcal{A}} : q \in states(\mathcal{A}) \mapsto AAct_{\mathcal{A}}(q) \cap in(\mathcal{A})(q)$ (adversary inputs) and (iv) $AO_{\mathcal{A}} : q \in states(\mathcal{A}) \mapsto AAct_{\mathcal{A}}(q) \cap out(\mathcal{A})(q)$ (adversary outputs).

Let $S_{\mathcal{A}} : q \in states(\mathcal{A}) \mapsto S_{\mathcal{A}}(q) \subseteq out(\mathcal{A})(q)$. We note $EAct_{\mathcal{A}} \setminus S_{\mathcal{A}} : q \in states(\mathcal{A}) \mapsto EAct_{\mathcal{A}}(q) \setminus S_{\mathcal{A}}(q))$. We note $hide((\mathcal{A}, EAct_{\mathcal{A}}), S_{\mathcal{A}}) = (hide(\mathcal{A}, S_{\mathcal{A}}), EAct_{\mathcal{A}} \setminus S_{\mathcal{A}})$

When this is clear in the context, we slightly abuse the notation and call a structured PSIOA a PSIOA.

Observe that nothing prevent us to require that $(\breve{EAct}_{\mathcal{A}}, \breve{AAct}_{\mathcal{A}})$ is a partition of $acts(\mathcal{A})$ s. t. an action a cannot be an environment action in a state and become an adversary action in another state.

We state the compatibility conditions and the composabilty operation for compatible structured PSIOA.

Definition 4.18 (Compatible structured PSIOA). Two structured PSIOA $(\mathcal{A}_1, EAct_{\mathcal{A}_1})$ and $(\mathcal{A}_2, EAct_{\mathcal{A}_2})$ are partially-compatible at state $(q_1, q_2) \in states(\mathcal{A}_1)\times states(\mathcal{A}_2)$ (resp. partially-compatible structured PSIOA) if $\mathcal{A}_1$ and $\mathcal{A}_2$ are partially-compatible at state $(q_1, q_2) \in states(\mathcal{A}_1) \times states(\mathcal{A}_2)$ (resp. partially-compatible PSIOA) and $\widehat{sig}(\mathcal{A}_1)(q_1)\cap\widehat{sig}(\mathcal{A}_2)(q_2) = EAct_{\mathcal{A}_1}(q_1)\cap EAct_{\mathcal{A}_2}(q_2)$ (resp. $\forall (q_1, q_2) \in states(\mathcal{A}_1||states(\mathcal{A}_2), \widehat{sig}(\mathcal{A}_1)(q_1)\cap\widehat{sig}(\mathcal{A}_2)(q_2)$ $= EAct_{\mathcal{A}_1}(q_1) \cap EAct_{\mathcal{A}_2}(q_2)$). That is, every shared action must be an environment action of both structured PSIOA.

Definition 4.19 (Structured PSIOA composition). Given partially-compatible structured PSIOA $(\mathcal{A}_1, EAct_{\mathcal{A}_1})$ and $(\mathcal{A}_2, EAct_{\mathcal{A}_2})$, their partial-composition $(\mathcal{A}_1, EAct_{\mathcal{A}_1})||(\mathcal{A}_2, EAct_{\mathcal{A}_2})$ is the structured PSIOA $(\mathcal{A}_1||\mathcal{A}_2, EAct_{\mathcal{A}_1} \cup EAct_{\mathcal{A}_2})$ where $EAct_{\mathcal{A}_1} \cup EAct_{\mathcal{A}_2} : (q_1, q_2) \in states(\mathcal{A}_1||\mathcal{A}_2) \mapsto EAct_{\mathcal{A}_1}(q_1) \cup EAct_{\mathcal{A}_2}(q_2)$.

We can also extend the previous definition to PCA:

Definition 4.20 (Structured configuration). A structured configuration is a pair $(\mathbf{A}, \mathbf{S})$ where $\mathbf{A}$ is a family of structured PSIOA and $\mathbf{S}$ is a mapping function with domain $\mathbf{A}$ such that for every $\mathcal{A} \in \mathbf{A}, \mathbf{S}(\mathcal{A}) \in states(\mathcal{A})$. Furthermore, in addition of attributes of definition 2.11, we note $EAct(C) = \bigcup_{\mathcal{A}\in\mathbf{A}} EAct_{\mathcal{A}}(\mathbf{S}(\mathcal{A}))$.

Definition 4.21 (Compatible structured configuration). A structured configuration $(\mathbf{A}, \mathbf{S})$ is compatible iff, for all $\mathcal{A}, \mathcal{B} \in \mathbf{A}, \mathcal{A} \neq \mathcal{B}$, $\mathcal{A}, \mathcal{B}$ are compatible at state $(\mathbf{S}(\mathcal{A}), \mathbf{S}(\mathcal{B}))$

Definition 4.22 (Structured PCA). A structured PCA X is a PCA s. t. (1) the attached PSIOA is replaced by a structured PSIOA,

(2) $config(X)$ is a function that maps every $q \in states(X)$ to a compatible structured configuration $config(X)(q)$ and (3) X is associated with the mapping $EAct_X$ with domain $states(X)$ s. t. $\forall q \in states(X)$, $EAct_X(q) = EAct_{psioa(X)}(q) = EAct(config(X)(q)) \setminus hidden\text{-}actions(X)(q)$

We can naturally define composition of partially-compatible structured PCA which is the same than the one of definition 4.19. Such a composition naturally yields to a structured PCA.

Lemma 4.23 (Closeness of structured PCA under composition). *Let X_1 and X_2 be partially-compatible structured PCA. Then $X_1||X_2$ is a structured PCA.*

Proof. See appendix C, lemma C.1 □

4.8 Adversary for structured automata

In the following we extend the notion of adversary introduced in [4] to adversary for structured PSIOA.

Definition 4.24 (Adversary for structured PSIOA). An adversary Adv for a structured PSIOA (resp. PCA) $(\mathcal{A}, EACT_{\mathcal{A}})$ is a PSIOA (resp. PCA) s. t.

- Adv is partially-compatible with $\mathcal{A}$
- For every $q = (q_{\mathcal{A}}, q_{Adv}) \in states(\mathcal{A}||Adv)$,
  - $IA_{\mathcal{A}}(q_{\mathcal{A}}) \subseteq out(Adv)(q_{Adv})$
  - $EAct_{\mathcal{A}}(q_{\mathcal{A}}) \cap \widehat{sig}(Adv)(q_{Adv}) = \emptyset$

We extend the definition to automata family: An adversary $\underline{Adv}$ for a structured PSIOA (resp. PCA) family $(\underline{\mathcal{A}}, \underline{EACT_{\mathcal{A}}}) = (\mathcal{A}_k, EACT_{\mathcal{A}_k})_{k\in\mathbb{N}}$ is a family $(Adv_k)_{k\in\mathbb{N}}$ of PSIOA (resp. PCA) s. t. $\forall k \in \mathbb{N}$, Adv_k is an adversary of $\mathcal{A}_k$

Lemma 4.25. *Suppose $\mathcal{A}$ and $\mathcal{B}$ are compatible structured PSIOA (resp. PCA), and Adv is an adversary for $\mathcal{A}||\mathcal{B}$. Then Adv is an adversary for $\mathcal{A}$. Also, if $\underline{\mathcal{A}}$ and $\underline{\mathcal{B}}$ are compatible structured PSIOA (resp. PCA) families, and $\underline{Adv}$ is an adversary family for $\underline{\mathcal{A}}||\underline{\mathcal{B}}$. Then $\underline{Adv}$ is an adversary family for $\underline{\mathcal{B}}$.*

Proof. Suppose $\mathcal{A}$ and $\mathcal{B}$ are compatible structured PSIOA and Adv is an adversary for $\mathcal{A}||\mathcal{B}$. We observe that the conditions of definition 4.24 are satisfied.

(1) Adv is compatible with $\mathcal{A}$. This follows from the fact that Adv is compatible with $\mathcal{A}||\mathcal{B}$.
(2) Let $q' = (q_{\mathcal{A}}, q_{Adv}) \in states(\mathcal{A}||Adv)$. Then it exists $q = (q_{\mathcal{A}}, q_{\mathcal{B}}, q_{Adv}) \in states(\mathcal{A}||\mathcal{B}||Adv)$. Since Adv is an adversary for $\mathcal{A}||\mathcal{B}$ we know that:
   - $IA_{\mathcal{A}}(q_{\mathcal{A}}) \cup IA_{\mathcal{B}}(q_{\mathcal{B}}) \subseteq out(Adv)(q_{Adv})$, which means that $IA_{\mathcal{A}}(q_{\mathcal{A}}) \subseteq out(Adv)(q_{Adv})$.
   - $EAct_{\mathcal{A}}(q_{\mathcal{A}}) \cup EAct_{\mathcal{B}}(q_{\mathcal{B}}) \cap \widehat{sig}(Adv)(q_{Adv}) = \emptyset$, which means that $EAct_{\mathcal{A}}(q_{\mathcal{A}}) \cap \widehat{sig}(Adv)(q_{Adv}) = \emptyset$.

The extension to structured PSIOA families and adversary families is straightforward. □

4.9 Dynamic Secure-Emulation

The framework is finally expressive enough to define secure-emulation [2] for distributed systems with (i) distributed scheduling and (ii) with potentially dynamic creation of automata. This relation will be shown to be (iii) composable, which is the main contribution of this paper.

Definition 4.26 (Secure Emulation). . Suppose $\underline{\mathcal{A}}$ and $\underline{\mathcal{B}}$ be structured PSIOA (resp. PCA) families. Let Sch be a scheduler schema and $f_{(.,.)}$ be an insight function.

We say that $\underline{\mathcal{A}}$ secure-emulates $\underline{\mathcal{B}}$ w.r.t. Sch and f (denoted $\underline{\mathcal{A}} \leq_{SE}^{Sch,f} \underline{\mathcal{B}}$) if, for every adversary family $\underline{Adv}$ for $\underline{\mathcal{A}}$ with polynomially bounded description, there is an adversary family $\underline{Sim}$ for $\underline{\mathcal{B}}$ with polynomially bounded description such that: $hide(\underline{\mathcal{A}}||\underline{Adv}, AAct_{\underline{\mathcal{A}}}) \leq_{neg,pt}^{Sch,f} hide(\underline{\mathcal{B}}||\underline{Sim}, AAct_{\underline{\mathcal{B}}})$. Transitivity of $\leq_{SE}^{Sch,f}$ follows immediately from transitivity of $\leq_{neg,pt}^{Sch,f}$.

Dummy Adversary. To prove composability of secure-emulation, we use the well-known technique, introduced by Canetti [2], based on dummy-adversary which plays the role of a forwarder between a structured PSIOA (resp. PCA) $\mathcal{A}$ and another (potentially more sophisticated) adversary of a $g(\mathcal{A})$ where g is an action-renaming function.

Let $\underline{\mathcal{A}}$ be a structured PSIOA family and, for each $k \in \mathbb{N}$, let g_k be a partial function defined on $states(\mathcal{A}) \times acts(\mathcal{A})$ s. t. $\forall q \in states(\mathcal{A})$, $g_k(q)$ is a bijection from $AAct_{\mathcal{A}_k}$ to a set of fresh action names. We refer to $\underline{g} = \{g_k\}_{k\in\mathbb{N}}$ as a renaming of adversary actions for $\mathcal{A}$, and we write $\underline{g}(\underline{\mathcal{A}})$ for the result of applying g_k to $\mathcal{A}_k$ for every $k \in \mathbb{N}$.

Definition 4.27 (Dummy Adversary). Let $\mathcal{A}$ be a PSIOA (resp. PCA) and g be a bijection from $AAct_{\mathcal{A}}$ to a set of fresh action names. Then $Dummy(\mathcal{A}, g)$ is the PSIOA (resp. PCA) Adv' defined as follows:

- (States) Every state q of Adv' is described by its unique variable $q.pending \in \overbrace{AO}_{\mathcal{A}} \cup f(\overbrace{AI}_{\mathcal{A}}) \cup \{\perp\}$
- (Start state) $start(Adv').pending = \perp$
- (Signature) $\forall q \in states(Adv')$,
  - $in(Adv')(q) = \overbrace{in}(Adv') = \overbrace{AO}_{\mathcal{A}} \cup g(\overbrace{AI}_{\mathcal{A}})$
  - $int(Adv')(q) = \emptyset$
  - $*$ $out(Adv')(q) = \{a\}$ if $g(a) = q.pending \in f(AI_{\mathcal{A}})$
    $*$ $out(Adv')(q) = \{g(a)\}$ if $a = q.pending \in AO_{\mathcal{A}}$
    $*$ $out(Adv')(q) = \emptyset$ if $q.pending = \perp$.
- (Transition) for every $q \in states(Adv')$, $\forall a \in \widehat{sig}(Adv')(q)$, $supp(\eta_{(Adv',q,a)}) = \{q'\}$ s. t.
  - if $a \in in(Adv')(q)$, $q'.pending = a$
  - if $a \in out(Adv')(q)$, $q'.pending = \perp$

We extend this definition for families: Let $\underline{\mathcal{A}}$ be a structured PSIOA (resp. PCA) family $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$. Let $\underline{g} = \{g_k\}_{k\in\mathbb{N}}$ be a family of bijection from $\overbrace{AAct}_{\mathcal{A}_k}$ to a set of fresh action names. Then $Dummy(\underline{\mathcal{A}}, \underline{g}) = \{Dummy(\mathcal{A}_k, g_k)\}_{k\in\mathbb{N}}$ is a dummy adversary family for $\underline{\mathcal{A}}$.

The following lemma 4.29 shows that dummy adversaries can be transparently added between a structured PSIOA (resp. PCA) and an adversary for that structured PSIOA. This fact is used in the proof of composability of secure-emulation, with the classic decomposition technique introduced by Canetti [2]. Additional work is required compared to [4] since we have to deal with a more general definition that enables schedulers that are not task-schedules. However, the approach follows the same methodology, i.e. when the scheduler $\underline{\sigma}$ instructs to trigger an action shared by $\underline{g}(\underline{\mathcal{A}})$ and $\underline{Adv}$, the corresponding balanced scheduler $\underline{\sigma'}$ successively orders to trigger

the corresponding action (modulo a potential renaming) and the attached forward by the dummy adversary. The proof of this lemma 4.29 introduces two natural constructions $Forward^{e}_{(\underline{\mathcal{A}},\underline{g},\underline{Adv})}$ and $Forward^{s}_{(\underline{\mathcal{A}},\underline{g},\underline{Adv})}$, to formalise the fact that the dummy adversary forwards the actions between $\underline{\mathcal{A}}$ and $\underline{Adv}$. The following properties should always be verified by reasonable pair made up of a scheduler schema and an insight function.

Definition 4.28 (brave forwarding construction). A pair (Sch, f) made up of a scheduler schema and an insight function is said *brave* if: For every structured PSIOA (resp. PCA) family $\underline{\mathcal{A}} = \{\mathcal{A}_k\}_{k\in\mathbb{N}}$, for every family of bijections $\underline{g} = \{g_k\}_{k\in\mathbb{N}}$ from $\breve{AAct}_{\mathcal{A}_k}$ to a set of fresh action names, for every $\underline{Adv} = \{Adv_k\}_{k\in\mathbb{N}}$ being an adversary for both $\underline{g}(\underline{\mathcal{A}})$ and $\underline{H} = hide(\underline{\mathcal{A}}||Dummy(\underline{\mathcal{A}},\underline{g}), AAct_{\underline{\mathcal{A}}})$, for every environment $\underline{\mathcal{E}} = (\mathcal{E}_k)_{k\in\mathbb{N}}$ of both $\underline{\Phi} = hide(\underline{g}(\underline{\mathcal{A}})||\underline{Adv}, AAct_{\underline{\mathcal{A}}})$ and $\underline{\Psi} = hide(\underline{H}||\underline{Adv}, AAct_{\underline{\mathcal{A}}})$ and $\forall k \in \mathbb{N}$:

- $f_{(\mathcal{E}_k,\Phi_k)}(\alpha) = f_{(\mathcal{E}_k,g_k(\mathcal{A}_k)||Adv_k)}(\alpha), \forall \alpha \in execs^*(\mathcal{E}_k||\Phi_k)$
- $f_{(\mathcal{E}_k,\Psi_k)}(\alpha') = f_{(\mathcal{E}_k,H_k||Adv_k)}(\alpha), \forall \alpha' \in execs^*(\mathcal{E}_k||\Psi_k)$
- $f_{(\mathcal{E}_k,\Phi_k)}(\alpha) = f_{(\mathcal{E}_k,\Psi_k)}(\alpha'), \forall \alpha \in execs^*(\mathcal{E}_k||\Phi_k), \forall \alpha' = Forward^{e}_{(\mathcal{A}_k,g_k,Adv_k)}(\alpha)$,
- $\forall \sigma \in Sch(\mathcal{E}_k||\Phi_k), Forward^{s}_{(\mathcal{A}_k,g_k,Adv_k)}(\sigma) \in Sch(\mathcal{E}_k||\Psi_k)$

Lemma 4.29 (Dummy adversary insertion). *Let (Sch, f) be a brave pair made of a scheduler schema Sch and an insight function f. Let $\underline{\mathcal{A}}$ be a structured PSIOA (resp. PCA) family $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$. Let $\underline{g} = \{g_k\}_{k\in\mathbb{N}}$ be a family of bijection from $\breve{AAct}_{\mathcal{A}_k}$ to a set of fresh action names. Let $\underline{Adv} = \{Adv_k\}_{k\in\mathbb{N}}$ be an adversary for both $\underline{g}(\underline{\mathcal{A}})$ and $hide(\underline{\mathcal{A}}||Dummy(\underline{\mathcal{A}},\underline{g}), AAct_{\underline{\mathcal{A}}})$. Then,*

$$\underline{g}(\underline{\mathcal{A}})||\underline{Adv} \leq^{Sch,f}_{neg,pt} hide(\underline{\mathcal{A}}||Dummy(\underline{\mathcal{A}},\underline{g}), AAct_{\underline{\mathcal{A}}})||\underline{Adv}$$

Proof. See appendix D, lemma D.1. □

We can use previous lemma 4.29 to use the technique of reduction to dummy adversary introduced by Canetti [2].

Theorem 4.30 (Composability of dynamic secure-emulation). *Let (Sch, f) be a brave pair made of a scheduler schema Sch and an insight function f. Let $b \in \mathbb{N}$. Let $\underline{\mathcal{A}}^1, \underline{\mathcal{A}}^2, ..., \underline{\mathcal{A}}^b$ and $\underline{\mathcal{B}}^1, \underline{\mathcal{B}}^2, ..., \underline{\mathcal{B}}^b$ be pair- wise partially-compatible polynomial-time-bounded structured PSIOA (resp. PCA) families, with $\underline{\mathcal{A}}^i \leq^{Sch,f}_{SE} \underline{\mathcal{B}}^i$ for every $i \in [1, b]$. Then, we have $\widehat{\mathcal{A}} \leq_{SE} \widehat{\mathcal{B}}$ with $\widehat{\mathcal{A}} = \underline{\mathcal{A}}^1||\underline{\mathcal{A}}^2||...||\underline{\mathcal{A}}^b$ and $\widehat{\mathcal{B}} = \underline{\mathcal{B}}^1||\underline{\mathcal{B}}^2||...||\underline{\mathcal{B}}^b$.*

Proof. Let $\underline{Adv}$ be an adversary family for $\widehat{\mathcal{A}}$ with polynomially bounded description. We need to construct an adversary family $\underline{Sim}$ for $\widehat{\mathcal{B}}$ with polynomially bounded description such that: $hide(\widehat{\mathcal{A}}||\underline{Adv}, AAct_{\widehat{\mathcal{A}}}) \leq^{Sch,f}_{neg,pt} hide(\widehat{\mathcal{B}}||\underline{Sim}, AAct_{\widehat{\mathcal{B}}})$.

For every $(i,k) \in [1,b] \times \mathbb{N}$. We note g^i_k an arbitrary bijection from $AAct_{\mathcal{A}^i_k}$ to a set of fresh action names, i. e. $\underline{g}^i = \{g^i_k\}_{k\in\mathbb{N}}$ is a renaming of adversary actions for $\underline{\mathcal{A}}^i$. These functions induce a renaming for $\underline{\widehat{\mathcal{A}}}$: $\underline{g} = \{g_k\}_{k\in\mathbb{N}}$ with $\forall k \in \mathbb{N}, g_k = g^1_k \cup ... \cup g^b_k$, i. e. $\forall q \in states(\underline{\widehat{\mathcal{A}}}_k), \forall a \in AAct_{\underline{\widehat{\mathcal{A}}}}(q), g_k(a) = g^i_k(a)$ iff $a \in AAct_{\mathcal{A}^i_k}(q \restriction \mathcal{A}^i_k)$. We recall that compatibility definition for structured PSIOA requires that shared actions of two automata cannot be a adversary actions of their composition.

Let $\widehat{\underline{Dum}} = Dummy(\underline{\mathcal{A}}^1, \underline{g}^1)||...Dummy(\underline{\mathcal{A}}^b, \underline{g}^b)$. Let $i \in [1, b]$. Since $\underline{\mathcal{A}}^i \leq_{SE} \underline{\mathcal{B}}^i$, then $\exists \underline{DSim}^i$ s. t. $hide(\underline{\mathcal{A}}^i||Dummy(\underline{\mathcal{A}}^i, \underline{g}^i), AAct^i_{\underline{\underline{\mathcal{A}}}})$ $\leq_{neg,pt} hide(\underline{\mathcal{B}}^i||\underline{DSim}^i, AAct^i_{\underline{\underline{\mathcal{B}}}})$.

We note $\widehat{\underline{DSim}} = \underline{DSim}^1||...||\underline{DSim}^b$.

We observe the following:

$hide(\widehat{\underline{\mathcal{A}}}||\underline{Adv}, AAct_{\widehat{\underline{\mathcal{A}}}}) \equiv^{Sch,f}_{neg,pt}$

$hide(\underline{g}(\widehat{\underline{\mathcal{A}}})||\underline{g}(\underline{Adv}), \underline{g}(AAct_{\widehat{\underline{\mathcal{A}}}})) \leq^{Sch,f}_{neg,pt}$

$hide(\widehat{\underline{\mathcal{A}}}||\widehat{\underline{Dum}}||\underline{g}(\underline{Adv}), \underline{g}(AAct_{\widehat{\underline{\mathcal{A}}}}) \cup AAct_{\widehat{\underline{\mathcal{A}}}}) \leq^{Sch,f}_{neg,pt}$

$hide(\widehat{\underline{\mathcal{B}}}||\widehat{\underline{DSim}}||\underline{g}(\underline{Adv}), \underline{g}(AAct_{\widehat{\underline{\mathcal{A}}}}) \cup AAct_{\widehat{\underline{\mathcal{B}}}}) \equiv^{Sch,f}_{neg,pt}$

$hide(\widehat{\underline{\mathcal{B}}}||hide(\widehat{\underline{DSim}}||\underline{g}(\underline{Adv}), \underline{g}(AAct_{\widehat{\underline{\mathcal{A}}}})), AAct_{\widehat{\underline{\mathcal{B}}}})$

Here, the first relation follows from the property of renaming, the second from lemma 4.29, the third from theorem 4.15, and the last from the properties of the hiding operator.

We define $\underline{Sim} = hide(\widehat{\underline{DSim}}||\underline{g}(\underline{Adv}), \underline{g}(AAct_{\widehat{\underline{\mathcal{A}}}}))$

Hence we have shown that for every adversary family $\underline{Adv}$ for $\widehat{\underline{\mathcal{A}}}$ with polynomially bounded description it exists a polynomially bounded adversary $\underline{Sim}$ for $\widehat{\underline{\mathcal{B}}}$ such that: $hide(\widehat{\underline{\mathcal{A}}}||\underline{Adv}, AAct_{\widehat{\underline{\mathcal{A}}}}) \leq^{Sch,f}_{neg,pt}$ $hide(\widehat{\underline{\mathcal{B}}}||\underline{Sim}, AAct_{\widehat{\underline{\mathcal{B}}}})$, which ends the proof.

□

5 CONCLUSION

In this paper we extended the *composable secure-emulation* of Canetti et al. [4] to dynamic settings on top of dynamic probabilistic I/O automata. The purpose of our extension is to provide a complete framework to model dynamic probabilistic systems using cryptographic modules. It should be noted that the universal composition [2] framework is the only framework used so far in order to model blockchain algorithmic building blocks (e.g. [8]). However, as discussed earlier in our paper, this framework has as major limitation the fact that it does not cover the dynamicity aspects under a distributed scheduling(see [5] for a detailed discussion related to the difference between the centralized and distributed schedulers). Our work is the first complete framework allowing to specify and prove the correctness of secure probabilistic distributed dynamic systems (e.g. blockchains) under distributed schedulers. The completeness of our framework comes from its ability to cover the distributed aspects of scheduling, the probabilistic nature of algorithms, the dynamicity of the system and the simulation based cryptography.

REFERENCES

[1] Paul C. Attie and Nancy A. Lynch. 2016. Dynamic input/output automata: A formal and compositional model for dynamic systems. *Inf. Comput.* 249 (2016), 28–75. https://doi.org/10.1016/j.ic.2016.03.008

[2] Ran Canetti. 2020. Universally Composable Security. *J. ACM* 67, 5 (2020), 28:1–28:94. https://doi.org/10.1145/3402457

[3] Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. 2018. Task-Structured Probabilistic {I/O} Automata. *J. Comput. System Sci.* 94 (2018), 63—-97. https://doi.org/10.1016/j.jcss.2017.09.007

[4] Ran Canetti, Ling Cheung, Dilsun Kirli Kaynar, Nancy A. Lynch, and Olivier Pereira. 2007. Compositional Security for Task-PIOAs. In *20th IEEE Computer*

Security Foundations Symposium, CSF 2007, 6-8 July 2007, Venice, Italy. IEEE Computer Society, 125–139. https://doi.org/10.1109/CSF.2007.15

[5] Ran Canetti, Ling Cheung, Nancy A. Lynch, and Olivier Pereira. 2007. On the Role of Scheduling in Simulation-Based Security. *IACR Cryptol. ePrint Arch.* (2007), 102. http://eprint.iacr.org/2007/102

[6] Jing Chen and Silvio Micali. 2019. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.* 777 (2019), 155–183. https://doi.org/10.1016/j.tcs.2019.02.001

[7] Pierre Civit and Maria Potop-Butucaru. 2021. Probabilistic Dynamic Input Output Automata. *IACR Cryptol. ePrint Arch.* (2021), 798. https://eprint.iacr.org/2021/798

[8] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 9057)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, 281–310. https://doi.org/10.1007/978-3-662-46803-6_10

[9] Maurice Herlihy. 2017. Blockchains and the Future of Distributed Computing. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, Elad Michael Schiller and Alexander A. Schwarzmann (Eds.). ACM, 155. https://doi.org/10.1145/3087801.3087873

[10] Dennis Hofheinz and Victor Shoup. 2015. GNUC: A New Universal Composability Framework. *J. Cryptol.* 28, 3 (2015), 423–508. https://doi.org/10.1007/s00145-013-9160-y

[11] Ralf Küsters, Max Tuengerthal, and Daniel Rausch. 2020. The IITM Model: A Simple and Expressive Model for Universal Composability. *J. Cryptol.* 33, 4 (2020), 1461–1584. https://doi.org/10.1007/s00145-020-09352-1

[12] Nancy Lynch, Michael Merritt, William Weihl, and Alan Fekete. 1988. A theory of atomic transactions. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 326 LNCS (1988), 41–71. https://doi.org/10.1007/3-540-50171-1_3

[13] Alejandro Ranchal-Pedrosa and Vincent Gramoli. 2019. Platypus: Offchain Protocol Without Synchrony. In *18th IEEE International Symposium on Network Computing and Applications, NCA 2019, Cambridge, MA, USA, September 26-28, 2019*, Aris Gkoulalas-Divanis, Mirco Marchetti, and Dimiter R. Avresky (Eds.). IEEE, 1–8. https://doi.org/10.1109/NCA.2019.8935037

[14] Roberto Segala. 1995. *Modeling and Verification of Randomized Distributed Real-Time Systems*. Ph.D. Dissertation. Massachusettes Institute of technology.

A RESULTS FOR FOUNDATIONAL LAYER

Lemma A.1 (PSIOA closeness under action-renaming). *Let $\mathcal{A}$ be a PSIOA and let ar be a partial function on $states(\mathcal{A})\times acts(\mathcal{A})$, s. t. $\forall q \in states(\mathcal{A})$, $ar(q)$ is an injective mapping with $\widehat{sig}(\mathcal{A})(q)$ as domain. Then $ar(\mathcal{A})$ is a PSIOA.*

Proof. We need to show (1) $\forall (q,a,\eta),(q,a,\eta') \in dtrans(\mathcal{A})$, $\eta = \eta'$ and $a \in \widehat{sig}(\mathcal{A})(q)$, (2) $\forall q \in states(\mathcal{A}), \forall a \in \widehat{sig}(\mathcal{A})(q)$, $\exists \eta \in Disc(states(\mathcal{A}))$, $(q,a,\eta) \in dtrans(\mathcal{A})$ and (3) $\forall q \in states(\mathcal{A})$: $in(\mathcal{A})(q) \cap out(\mathcal{A})(s) = in(\mathcal{A})(q) \cap int(\mathcal{A})(q) = out(\mathcal{A})(q) \cap int(\mathcal{A})(q) = \emptyset$.

- Constraint 1: From definition 2.8, we have, for any $q \in states(ar(\mathcal{A}))$: $\widehat{sig}(ar(\mathcal{A}))(q) = out(ar(\mathcal{A}))(q) \cup in(ar(\mathcal{A}))(q) \cup int(ar(\mathcal{A}))(q) = ar(out(\mathcal{A})(q)) \cup ar(in(\mathcal{A})(q)) \cup ar(int(\mathcal{A})(q)) = ar(\widehat{sig}(\mathcal{A})(q))$. Since $\mathcal{A}$ is a PSIOA, we have $\forall (q,a,\eta),(q,a,\eta') \in dtrans(\mathcal{A}) : a \in \widehat{sig}(\mathcal{A})(q)$ and $\eta = \eta'$. From definition 2.8, $dtrans(ar(\mathcal{A})) = \{(q, ar(a), \eta) | (q,a,\eta) \in dtrans(\mathcal{A})\}$ Hence, if $(q, ar(a), \eta)$, $(q, ar(a), \eta')$ are arbitrary element of $dtrans(ar(\mathcal{A}))$, then $(q,a,\eta),(q,a,\eta') \in dtrans(\mathcal{A})$, and so $\eta = \eta'$ and $a \in \widehat{sig}(\mathcal{A})(q)$. Hence $ar(a) \in ar(\widehat{sig}(\mathcal{A})(q))$. Since $ar(\widehat{sig}(\mathcal{A})(q)) = \widehat{sig}(ar(\mathcal{A}))(q)$, we conclude $ar(a) \in \widehat{sig}(ar(\mathcal{A}))(q)$. Hence, $\forall (q, ar(a), \eta)$, $(q, ar(a), \eta') \in dtrans\,(ar(\mathcal{A})) : ar(a) \in \widehat{sig}(ar(\mathcal{A}))(q)$and $\eta = \eta'$. Thus, Constraint 1 holds for $ar(\mathcal{A})$.
- Constraint 2: From definition 2.8, $states(ra(\mathcal{A})) = states(\mathcal{A})$, $dtrans(ra(\mathcal{A})) = (q, ra(a), \eta) | (q,a,\eta) \in dtrans(\mathcal{A})$, and for all $q \in states(ra(\mathcal{A}))$, $in(ra(\mathcal{A}))(q) = ra(in(\mathcal{A})(q))$. Let q be any state of $ra(\mathcal{A})$, and let $q \in \widehat{sig}(ra(\mathcal{A}))(q)$. Then $b = ra(a)$ for some $a \in \widehat{sig}(\mathcal{A})(q)$. We have $(q,a,\eta) \in dtrans(\mathcal{A})$ for some η, by Constraint 2 of action enabling for $\mathcal{A}$. Hence $(q,a,\eta) \in dtrans(ra(\mathcal{A}))$. Hence $(q,b,\eta) \in dtrans(ra(\mathcal{A}))$. Hence Constraint 2 holds for $ra(\mathcal{A})$.
- Constraint 3: $\mathcal{A}$ is a PSIOA and so satisfies Constraint 3. From this and definition 2.8 and the requirement that ra be injective, it is easy to see that $ra(\mathcal{A})$ also satisfies Constraint 3.

□

B RESULTS FOR BOUNDED PSIOA

Lemma B.1 (composition of bounded PSIOA is a bounded PSIOA). *There exists a constant c_{comp} such that the following holds. Suppose $\mathcal{A}_1$ is a b_1-time-bounded PSIOA and $\mathcal{A}_2$ is a b_2-time-bounded PSIOA, where $b_1, b_2 \geq 1$. Then $\mathcal{A}_1||\mathcal{A}_2$ is a $c_{comp}\cdot(b_1+b_2)$-bounded PSIOA.*

Proof. We describe how the different bounds of definition 4.1 combine when we compose $\mathcal{A}_1$ and $\mathcal{A}_2$.

(1) Automaton parts: Every action has a standard representation which is the same as its representation in $\mathcal{A}_1$ or $\mathcal{A}_2$. The length of this representation is, therefore, at most $max(b_1, b_2)$. Every state of $\mathcal{A}_1||\mathcal{A}_2$ can be represented with a $2\cdot(b_1+b_2)+2 \leq 3\cdot(b_1+b_2)$-bit string, by following each bit of the bit-string representations of the states of $\mathcal{A}_1$ and $\mathcal{A}_2$ with a zero, and then concatenating the results, separating them with the string 11. Likewise, every transition of $\mathcal{A}_1||\mathcal{A}_2$ can be represented as a $3\cdot(b_1+b_2)$-bit string, by combining the representations of transitions of one or both of the component automata.
(2) Decoding: It is possible to decide whether a candidate state $q = (q_1, q_2)$ is the start state of $\mathcal{A}_1||\mathcal{A}_2$ by checking if q_1 is the start state of $\mathcal{A}_1$ and q_2 is the start state of $\mathcal{A}_2$. Given the representation $\langle (q_1,q_2)\rangle$ of a state $(q_1,q_2) \in states(\mathcal{A}_1||\mathcal{A}_2)$, it is possible to decide if a candidate input action is an element of $in(\mathcal{A}_1||\mathcal{A}_2)(q_1,q_2)$ by checking if it is an element of $in(\mathcal{A}_1)(q_1)$ or $in(\mathcal{A}_2)(q_2)$ but not an element of $in(\mathcal{A}_1)(q_1)$ or $in(\mathcal{A}_2)(q_2)$. Given the representation $\langle (q_1,q_2)\rangle$ of a state $(q_1,q_2) \in states(\mathcal{A}_1||\mathcal{A}_2)$, it is possible to decide if a candidate action is an element of $out(\mathcal{A}_1||\mathcal{A}_2)(q_1,q_2)$ (resp. $int(\mathcal{A}_1||\mathcal{A}_2)(q_1,q_2)$) by checking if it is either an element of $out(\mathcal{A}_1)(q_1)$ or $out(\mathcal{A}_2)(q_2)$ (resp. $int(\mathcal{A}_1)(q_1)$ or $int(\mathcal{A}_2)(q_2)$) All these verifications can be done in time $O(b_1+b_2)$. Given the representations $\langle (q_1,q_2)\rangle$ and $\langle a\rangle$ of a state $(q_1,q_2) \in states(\mathcal{A}_1||\mathcal{A}_2)$ and an action $a \in \widehat{sig}(\mathcal{A}_1||\mathcal{A}_2)((q_1,q_2))$, it is possible to decide whether a candidate transition $tr = ((q_1,q_2),a,\eta_1\otimes\eta_2)$ is a transition of $\mathcal{A}_1||\mathcal{A}_2$ by checking if $tr_1 = (q_1,a,\eta_1)$ is a transition of $\mathcal{A}_1$ or $tr_2 = (q_2,a,\eta_2)$ is a transition of $\mathcal{A}_2$ after having extracted the bit-string representation of q_1, q_2, tr_1, tr_2 with time $O(b_1+b_2)$.
(3) Determining the next state: Assume M_{state1} and M_{state2} are the probabilistic Turing machines described in last item of definition 4.1 for $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. We define M_{state} for $\mathcal{A}_1||\mathcal{A}_2$ as the probabilistic Turing machine that, given state $q = (q_1,q_2)$ of $\mathcal{A}_1||\mathcal{A}_2$ where $q_1 = q \upharpoonright \mathcal{A}_1$ and

$q_2 = q \upharpoonright \mathcal{A}_2$ and action $a \in \widehat{sig}(\mathcal{A})(q)$, outputs the next state of $\mathcal{A}_1||\mathcal{A}_2$ as $q' = (q'_1, q'_2)$, where q'_1 is the next state of $\mathcal{A}_1$ and q'_2 is the next state of $\mathcal{A}_2$. The state q' is computed as follows: If $a \in \widehat{sig}(\mathcal{A}_1)(q_1)$, then q'_1 is the output of $M_{state1}(q_1, a)$, while $q'_1 = q_1$ otherwise. If $a \in \widehat{sig}(\mathcal{A}_2)(q_2)$ then q'_2 is the output of $M_{state2}(q_2, a)$, while $q'_2 = q_2$ otherwise. M_{state} always operates within time $O(b_1 + b_2)$: this time is sufficient to determine whether $a \in \widehat{sig}(\mathcal{A}_1)(q_1)$ and/or $a \in \widehat{sig}(\mathcal{A}_2)(q_2)$, to extract the needed parts of q to run M_{state1} and/or M_{state2}. Using standard Turing machine encoding, each of the needed Turing machines can be represented using $O(b_1 + b_2)$ bits.

□

In the following we extend the previous result to the PCA composition.

Lemma B.2 (composition of bounded PCA is a bounded PCA). *There exists a constant c'_{comp} such that the following holds. Suppose X_1 is a b_1-time-bounded PCA and X_2 is a b_2-time-bounded PCA, where $b_1, b_2 \geq 1$. Then $X_1||X_2$ is a $c'_{comp} \cdot (b_1 + b_2)$-bounded PCA.*

Proof.
- $psioa(X_1||X_2) = psioa(X_1)||psioa(X_2)$ which implies $psioa(X_1||X_2)$ is a $c \cdot (b_1 + b_2)$-bounded PSIOA.
- it is sufficient to reserve a special constant-sized sequence of bits b^* for concatenation and hence, $\forall (q_1, q_2) \in states(X_1||X_2)$, $\forall a \in \widehat{sig}(X_1||X_2)(q_1, q_2)$, obtaining a length of $O(b_1 + b_2)$ bits for the representation of $config(X_1||X_2)(q_1, q_2)$, $hidden\text{-}actions(X_1||X_2)(q_1, q_2)$, $created(X_1||X_2)(q_1, q_2)(a)$.
- given the representation $\langle q \rangle$ of $q = (q_1, q_2) \in states(X_1||X_2)$, the representation $\langle a \rangle$ of $a \in \widehat{sig}(X_1||X_2)((q_1, q_2))(a)$, it is sufficient to extract the representation $\langle q_1 \rangle$ of $q \upharpoonright \mathcal{A}_1$, extract the representation $\langle q_2 \rangle$ of $q \upharpoonright \mathcal{A}_2$, check if $a \in \widehat{sig}(\mathcal{A}_1)(q_1)$ (via $M_{sig1}(\langle q_1 \rangle, \langle a \rangle)$), check if $a \in \widehat{sig}(\mathcal{A}_2)(q_2)$ (via $M_{sig2}(\langle q_2 \rangle, \langle a \rangle)$), compute $\langle C_1 \rangle := M_{conf1}(\langle q_1 \rangle)$, $\langle C_2 \rangle := M_{conf2}(\langle q_2 \rangle)$, $\langle h_1 \rangle := M_{hidden1}(\langle q_1 \rangle)$, $\langle h_2 \rangle := M_{hidden2}(\langle q_2 \rangle)$, $\langle \varphi_1 \rangle := M_{created1}(\langle q_1 \rangle, \langle a \rangle)$, $\langle \varphi_2 \rangle := M_{created2}(\langle q_2 \rangle, \langle a \rangle)$ and finally perform a concatenation operation (with b^*) on respective pairs $(\langle C_1 \rangle, \langle C_2 \rangle)$, $(\langle h_1 \rangle, \langle h_2 \rangle)$, $(\langle \varphi_1 \rangle, \langle \varphi_2 \rangle)$. The computations are performed in time $O(b_1 + b_2)$. Using standard Turing machine encoding, each of the needed Turing machines can be represented using $O(b_1 + b_2)$ bits.

□

Lemma B.3 (hiding of bounded automata is bounded). *There exists a constant c_{hide} such that the following holds. Suppose $\mathcal{A}$ is a b-time-bounded PSIOA (resp. PCA), where $b \in \mathbb{R}^{\geq 0}$, $b \geq 1$. Let S be a b'-time recognizable function with $states(\mathcal{A})$ as domain. Then $hide(\mathcal{A}, S)$ is a $c_{hide} \cdot (b + b')$-time-bounded PSIOA (resp. PCA).*

Proof. All properties for $\mathcal{B} = hide(\mathcal{A}, S)$ are straightforward to check, except for the output actions and the internal actions. Let $\langle q \rangle$ be the bit-string representation of $q \in states(\mathcal{B})$. Let $\langle a \rangle$ be the bit-string representation of a candidate action a.

1. Output actions: To check whether a is an element of $out(\mathcal{B})(q)$, we use the fact that a is an element of $out(\mathcal{B})(q)$ if and only if a is an element of $out(\mathcal{A})(q)$ and is not in $S(q)$. So, to determine whether a is an element of $out(\mathcal{B})(q)$, we can use the procedure for checking whether a is an element of $out(\mathcal{A})(q)$, followed by checking whether a is in $S(q)$.

2. Internal actions: To check whether a is an element of $int(\mathcal{B})(q)$, we use the fact that a is an element of $int(\mathcal{B})(q)$ if and only if a is an element of $int(\mathcal{A})(q)$ or is in $S(q)$. So, to determine whether a is an element of $int(\mathcal{B})(q)$, we can use the procedure for checking whether a is an element of $int(\mathcal{A})(q)$, followed by checking whether a is in $S(q)$.

3. If $\mathcal{A}$ is a PCA, $hidden\text{-}actions(\mathcal{B})(q) = hidden\text{-}actions(\mathcal{A})(q) \cup S(q)$. By using a reserved special constant-sized sequence of bits b^* for concatenation, it the bit-string representation of $hidden\text{-}actions(\mathcal{B})(q)$ can easily have a size of $O(b + b')$ bits.

In all cases, the total time is proportional to $b+b'$. Using standard Turing machine encodings, each of the needed Turing machines can be represented using a number of bits that is proportional to $b + b'$. □

Theorem B.4 (Implementation transitivity). *Let Sch be a scheduler schema. Let $\epsilon_{12}, \epsilon_{23}\epsilon_{13} \in \mathbb{R}^{\leq 0}, p, q_1, q_2, q_3 \in \mathbb{N}$ with $\epsilon_{13} = \epsilon_{12}+\epsilon_{23}$, Let $f_{(\_,\_)}$ be an insight-function. Let $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ be PSIOA, s.t. $\mathcal{A}_1 \leq^{Sch,f}_{p,q_1,q_2,\epsilon_{12}} \mathcal{A}_2$ and $\mathcal{A}_2 \leq^{Sch,f}_{p,q_2,q_3,\epsilon_{23}} \mathcal{A}_3$, then $\mathcal{A}_1 \leq^{Sch,f}_{p,q_1,q_3,\epsilon_{13}} \mathcal{A}_3$.*

Proof. Let $\mathcal{E} \in env(\mathcal{A}_1) \cap env(\mathcal{A}_3)$ be p-bounded.

Case 1: $\mathcal{E} \in env(\mathcal{A}_2)$. Let $\sigma_1 \in Sch(\mathcal{E}||\mathcal{A}_1)$ q_1-bounded, then, since $\mathcal{A}_1 \leq^{Sch,f}_{p,q_1,q_2,\epsilon_{12}} \mathcal{A}_2$ it exists $\sigma_2 \in Sch(\mathcal{E}||\mathcal{A}_2)$ q_2-bounded s.t. $\sigma_1 S^{\leq \epsilon_{12}}_{\mathcal{E},f} \sigma_2$. and since $\mathcal{A}_2 \leq^{Sch,f}_{p,q_2,q_3,\epsilon_{23}} \mathcal{A}_3$, it exists $\sigma_3 \in Sch(\mathcal{E}||\mathcal{A}_3)$ q_3-bounded s.t. $\sigma_2 S^{\leq \epsilon_{23}}_{\mathcal{E},f} \sigma_3$ and so for every $\sigma_1 \in Sch(\mathcal{E}||\mathcal{A}_1)$ q_1-bounded, it exists $\sigma_3 \in Sch(\mathcal{E}||\mathcal{A}_3)$ q_3-bounded s.t. $\sigma_1 S^{\leq \epsilon_{13}}_{\mathcal{E},f} \sigma_3$, i. e. $\mathcal{A}_1 \leq^{Sch,f}_{p,q_1,q_3,\epsilon_{13}} \mathcal{A}_3$.

Case 2: $\mathcal{E} \notin env(\mathcal{A}_2)$. A renaming procedure has to be performed before applying Case 1.

Let $\mathbf{A} = \{\mathcal{E}, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}$. We note $acts(\mathbf{A}) = \bigcup_{\mathcal{B} \in \mathbf{A}} acts(\mathcal{B})$. We use the special character ® for our renaming which is assumed to not be present in any syntactical representation of any action in $acts(\mathbf{A})$.

We note ar_{int} the action renaming fction s. t. $\forall q \in \mathcal{E}$, $\forall a \in \widehat{sig}(\mathcal{E})(q)$, if $a \in int(\mathcal{E})(q)$, then $ar_{int}(q)(a) = a_{®int}$ and $ar_{int}(q)(a) = a$ otherwise. Then we note $\mathcal{E}' = ar_{int}(\mathcal{E})$.

If $\mathcal{E}'$ and $\mathcal{A}_2$ are not partially-compatible, it is only because of some reachable state $(q_{\mathcal{E}}, q_{\mathcal{A}_2}) \in states(\mathcal{E}') \times states(\mathcal{A}_2)$ s. t. $out(\mathcal{A}_2)(q_{\mathcal{A}_2}) \cap out(\mathcal{E}')(q_{\mathcal{E}}) \neq \emptyset$. Thus, we rename the actions for each state to avoid this conflict.

We note ar_{out} the renaming function for $\mathcal{E}'$, s. t. $\forall q_{\mathcal{E}} \in states(\mathcal{E})$, $\forall a \in \widehat{sig}(\mathcal{E})(q_{\mathcal{E}})$, $ar_{out}(q_{\mathcal{E}})(a) = a_{®out}$ if $a \in out(\mathcal{E})(q_{\mathcal{E}})$ and a otherwise. In the same way, We note, for every $i \in \{1, 2, 3\}$ ar^i_{in} the renaming function for $\mathcal{A}_i$, s. t. $\forall q_{\mathcal{A}_i} \in states(\mathcal{A}_i)$, $\forall a \in \widehat{sig}(\mathcal{A}_i)(q_{\mathcal{A}_i})$ $ar_{in}(q_{\mathcal{A}_i})(a) = a_{®out}$ if $a \in in(\mathcal{A}_i)(q_{\mathcal{A}_i})$ and a otherwise. Finally, $\mathcal{E}'' = ar_{out}(\mathcal{E}')$ and $\mathcal{A}''_i = ar^i_{in}(\mathcal{A}_i)$ are obviously partially-compatible (and even compatible) for each $i \in \{1, 2, 3\}$.

There is an obvious isomorphism between $\mathcal{E}''||\mathcal{A}''_1$ and $\mathcal{E}||\mathcal{A}_1$ and between $\mathcal{E}''||\mathcal{A}''_3$ and $\mathcal{E}||\mathcal{A}_3$ that allows us to apply case 1, which ends the proof.

□

Lemma B.5 (composability $\leq^{Sch,f}_{\underline{p},\underline{q_1},\underline{q_2},\underline{\epsilon}}$). *Let $\epsilon : \mathbb{N} \to \mathbb{R}^{\geq 0}$ and $p, p_3, q_1, q_2 : \mathbb{N} \to \mathbb{N}$ be given. Let Sch be a scheduler schema. Let $f_{(.,.)}$ be an insight function stable by composition. Let $\mathcal{A}$, $\mathcal{B}$ and C be 3 PSIOA (resp. PCA) families satisfying: C has p_3-bounded description and is partially compatible with both $\mathcal{A}$ and $\mathcal{B}$. Let c_{comp} be the constant factor associated with description bounds in parallel composition (see lemma 4.3) Then the following holds.*

If $\mathcal{A} \leq^{Sch,f}_{c_{comp}(p+p_3),q_1,q_2,\epsilon} \mathcal{B}$, then $C||\mathcal{A} \leq^{Sch,f}_{p,q_1,q_2,\epsilon} C||\mathcal{B}$.

Proof. Fix $\underline{\mathcal{A}} = (\mathcal{A}_k)_{k\in\mathbb{N}}$, $\underline{\mathcal{B}} = (\mathcal{B}_k)_{k\in\mathbb{N}}$, $\underline{C} = (C_k)_{k\in\mathbb{N}}$ and all the functions as in the hypotheses.

By definition 4.12, for every $k \in \mathbb{N}$, $\mathcal{A}_k \leq^{Sch,f}_{p'(k),q_1(k),q_2(k)} \mathcal{B}_k$ with $p' = (c_{comp} \cdot (p + p_3))$,

Thus, $\forall k \in \mathbb{N}$, $(C_k||\mathcal{A}_k) \leq^{Sch,f}_{p(k),q_1(k),q_2(k)} (C_k||\mathcal{B}_k)$ by lemma 4.13.

Finally, we obtain that $\underline{C}||\underline{\mathcal{A}} \leq_{Sch,p,q_1,q_2,\epsilon} \underline{C}||\underline{\mathcal{B}}$, as needed, applying definition 4.12 once again. □

Theorem B.6 (composability $\leq^{Sch,f}_{neg,pt}$). *Let Sch be a scheduler schema. Let $f_{(.,.)}$ be an insight function stable by composition.*

Let $\underline{\mathcal{A}_1}$, $\underline{\mathcal{A}_2}$ and $\underline{\mathcal{A}_3}$ be PSIOAs (resp. PCA) families satisfying: $\underline{\mathcal{A}_3}$ has p_3-bounded description where p_3 is a polynomial and is partially compatible with both $\underline{\mathcal{A}_1}$ and $\underline{\mathcal{A}_2}$. Then the following holds.

If $\underline{\mathcal{A}_1} \leq^{Sch,f}_{neg,pt} \underline{\mathcal{A}_2}$, then $\underline{\mathcal{A}_3}||\underline{\mathcal{A}_1} \leq^{Sch,f}_{neg,pt} \underline{\mathcal{A}_3}||\underline{\mathcal{A}_2}$. Observe that, by induction, Theorem generalizes to any constant number of substitutions.

Proof. Suppose $\underline{\mathcal{A}_1}$, $\underline{\mathcal{A}_2}$, $\underline{\mathcal{A}_3}$ and all the functions as in the hypotheses. Fix polynomial p_3 such that $\underline{\mathcal{A}_3}$ is p_3-time-bounded. To show that $\underline{\mathcal{A}_3}||\underline{\mathcal{A}_2} \leq^{Sch,f}_{neg,pt} \underline{\mathcal{A}_3}||\underline{\mathcal{A}_2}$, we fix polynomials p and q_1; we must obtain a polynomial q_2 and a negligible function ϵ such that $\underline{\mathcal{A}_3}||\underline{\mathcal{A}_2} \leq^{Sch,f}_{p,q_1,q_2,\epsilon} \underline{\mathcal{A}_3}||\underline{\mathcal{A}_2}$. Define p' to be the polynomial $c_{comp}(p + p_3)$. Since $\underline{\mathcal{A}_2} \leq^{Sch,f}_{neg,pt} \underline{\mathcal{A}_2}$, there exist a polynomial q_2 and a negligible function ϵ such that $\underline{\mathcal{A}_2} \leq^{Sch,f}_{p',q_1,q_2,\epsilon} \underline{\mathcal{A}_2}$. Lemma 4.14 then implies that $\underline{\mathcal{A}_3}||\underline{\mathcal{A}_2} \leq^{Sch,f}_{p',q_1,q_2,\epsilon} \underline{\mathcal{A}_3}||\underline{\mathcal{A}_2}$, as needed. □

C RESULTS FOR STRUCTURED AUTOMATA

Lemma C.1 (Closeness of structured PCA under composition). *Let X_1 and X_2 be partially-compatible structured PCA. Then $X_1||X_2$ is a structured PCA.*

Proof. Let X_1 and X_2 be partially-compatible structured PCA. Let $q_X = (q_{X_1}, q_{X_2}) \in states((X_1, X_2))$ We note $C_1 = config(X_1)(q_{X_1})$, $C_2 = config(X_2)(q_{X_2}), C = C_1 \cup C_2, h_1 = hidden\text{-}actions(X_1)(q_{X_1})$, $C_2 = hidden\text{-}actions(X_2)(q_{X_2})$, $h = h_1 \cup h_2$. The closeness under composition is ensured if the new restriction $EAct_X(q_X) = EAct(C)\backslash h$ is still ensured after composition, that is if $EAct_{X_1}(q_{X_1}) \cup EAct_{X_2}(q_{X_2}) = (EAct(C_1) \setminus h_1) \cup (EAct(C_2) \setminus h_2) = EAct(C_1 \cup C_2) \setminus (h_1 \cup h_2)$. This constraint is ensured for the same reason that the fourth one. Indeed, since X_1 and X_2 are partially-compatible by assumption, (i) the signatures $sig(config(X_1)\ (q_{X_1}))$ and the signature $sig(config(X_2)(q_{X_2}))$ are compatible and (ii) $sig(X_1)(q_{X_1})$ and $sig(X_2)(q_{X_2})$ are compatible. The conjonction of (i) and (ii) implies $h_1 \cap \widehat{sig}(X_2)(q_{X_2}) = h_2\ (q_{X_2}) \cap \widehat{sig}(X_1)(q_{X_1}) = \emptyset$. This is enough to ensure $(EAct(C_1) \setminus h_1) \cup (EAct(C_2) \setminus h_2) = EAct(C_1 \cup C_2) \setminus (h_1 \cup h_2)$. □

D RESULTS FOR ADVERSARIES

Lemma D.1 (Dummy adversary insertion). *Let (Sch, f) be a brave pair made of a scheduler schema Sch and an insight function f. Let $\underline{\mathcal{A}}$ be a structured PSIOA (resp. PCA) family $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$. Let $\underline{g} = \{g_k\}_{k\in\mathbb{N}}$ be a family of bijection from $A\widetilde{Act}_{\mathcal{A}_k}$ to a set of fresh action names. Let $\underline{Adv} = \{Adv_k\}_{k\in\mathbb{N}}$ be an adversary for both $\underline{g}(\underline{\mathcal{A}})$ and $hide(\underline{\mathcal{A}}||Dummy(\underline{\mathcal{A}}, \underline{g}), AAct_{\underline{\mathcal{A}}})$. Then,*

$$\underline{g}(\underline{\mathcal{A}})||\underline{Adv} \leq^{Sch,f}_{neg,pt} hide(\underline{\mathcal{A}}||Dummy(\underline{\mathcal{A}}, \underline{g}), AAct_{\underline{\mathcal{A}}})||\underline{Adv}$$

Proof. Let q_1 be any polynomial and set $q_2 := 2 \cdot q_1$. Let p, q be any polynomials and ϵ be the constant zero function, i. e. $\forall k \in \mathbb{N}, \epsilon(k) = 0$. Fix $k \in \mathbb{N}$, we note $D_k = Dummy(\mathcal{A}_k, g_k)$, $H_k = hide(\mathcal{A}_k||D_k, AAct_{\mathcal{A}_k})$ and $\underline{D} = (D_k)_{k\in\mathbb{N}}$ and $\underline{H} = (H_k)_{k\in\mathbb{N}}$. Let $\mathcal{E}$ be an environment for $g_k(\mathcal{A}_k)||Adv_k$ and for $\mathcal{A}_k||H_k||Adv_k$. Let $\sigma \in Sch(\mathcal{E}||g_k(\mathcal{A}_k)||Adv_k)$ be a q_1 bounded scheduler.

We are going to construct $\sigma' \in Sch(\mathcal{E}||H_k||Adv_k)$ balanced with σ and q_2 bounded scheduler in the intuitive way.

First we partition the functions depending if they are triggered by the environment or by the adversary. Hence, $\forall q = (q_{\mathcal{E}}, q_{\mathcal{A}}, q_{Adv}) \in states(\mathcal{E}||g_k(\mathcal{A}_k)\ ||\ Adv_k)$, for every $q^+ = (q_{\mathcal{E}}, q_{\mathcal{A}}, q_D, q_{Adv}) \in states(\mathcal{E}||H_k||Adv_k)$, we note:

- $E(q) = E(q^+) = \widehat{sig}(\mathcal{E}||g_k(\mathcal{A}_k)||Adv_k)((q_{\mathcal{E}}, q_{\mathcal{A}}, q_{Adv})) \setminus ([\widehat{ext}(g_k(\mathcal{A}_k))(q_{\mathcal{A}}) \cap \widehat{ext}(Adv_k)(q_{Adv})] \cup \widehat{ext}(D_k)(q_D))$, i. e. the actions not dedicated to the dummy adversary.
- – $F_{\mathcal{A}}(q) = [out(g_k(\mathcal{A}_k))(q_{\mathcal{A}}) \cap in(Adv_k)(q_{Adv})] \cap in(D_k)(q_D)$,
  – $F_{Adv}(q) = [in(g_k(\mathcal{A}_k))(q_{\mathcal{A}}) \cap out(Adv_k)(q_{Adv})] \cap in(D_k)(q_D)$
  – $F(q) = F_{\mathcal{A}}(q) \cup F_{Adv}(q)$
  – $F^+_{\mathcal{A}}(q^+) = in(D_k)(q_D) \cap out(\mathcal{A}_k)(q_{\mathcal{A}})$
  – $F^+_{Adv}(q^+) = in(D_k)(q_D) \cap out(Adv)(q_{Adv})$
  – $F^+(q) = F^+_{\mathcal{A}}(q) \cup F^+_{Adv}(q)$
  each set holds actions that have to be forwarded by the dummy adversary.
- $\forall a \in F^+_{\mathcal{A}}(q^+), origin(a) = g_k(a), forward(a) = g_k(a). \forall g_k(b) \in F^+_{Adv}(q^+), origin(g_k(b)) = g_k(b)$ and $forward(g_k(b)) = b$. When an action a is received by the dummy adversary, $origin(a)$ returns the corresponding action shared by $g_k(\mathcal{A}_k)$ and Adv_k, while $forward(a)$ returns the action forwarded by the dummy adversary with potential g_k-renaming.
- $G^+(q^+) = \widehat{sig}(\mathcal{E}||\mathcal{A}_k)||D_k||Adv_k)((q_{\mathcal{E}}, q_{\mathcal{A}}, q_D q_{Adv}))\backslash[E^+(q^+) \cup F^+(q^+)]$. These actions corresponds to a scenario where a new action is received by the dummy adversary before the appropriate forward. These actions will never by triggered by $\underline{\sigma'}$.

Now, we define a relationship between executions, noted $\alpha \sim \alpha'$, that captures the fact that the latter member α' of the relation corresponds to former one α when each action shared by $g_k(\mathcal{A}_k)$ and Adv_k is correctly forwarded by dummy adversary in α'.

$\forall(\alpha, \alpha') \in frags^*(\mathcal{E}||g_k(\mathcal{A}_k)||Adv_k) \times frags^*(\mathcal{E}||H_k||Adv_k)$ we note $\alpha \sim \alpha'$ iff:

- (initialisation): $\alpha' = start(\mathcal{E}||H_k, AAct_{\mathcal{A}_k})||Adv_k)$ and $\alpha = start(\mathcal{E}||g(\mathcal{A}_k)||Adv_k)$
- (environment side) $\alpha = (q_\mathcal{E}, q_\mathcal{A}, q_{Adv})\ a\ (q'_\mathcal{E}, q'_\mathcal{A}, q'_{Adv})$ and $\alpha' = (q_\mathcal{E}, q_\mathcal{A}, q_D, q_{Adv})\ a\ (q'_\mathcal{E}, q'_\mathcal{A}, q'_D, q'_{Adv})$ with $a \in E((q_\mathcal{E}, q_\mathcal{A}, q_{Adv}))$
- (forward) $\alpha = (q_\mathcal{E}, q_\mathcal{A}, q_{Adv})\ a\ (q_\mathcal{E}, q'_\mathcal{A}, q'_{Adv})$ and $\alpha' = (q_\mathcal{E}, q_\mathcal{A}, q_D, q_{Adv})\ b\ (q_\mathcal{E}, q'_\mathcal{A}, q'_D, q'_{Adv})\ b'\ (q_\mathcal{E}, q''_\mathcal{A}, q''_D, q''_{Adv})$ with $a = origin(b) \in F((q_\mathcal{E}, q_\mathcal{A}, q_{Adv}))$, $b' = forward(b)$ and $b \in F^+((q_\mathcal{E}, q_\mathcal{A}, q_D, q_{Adv}))$, .
- (generalization) $\alpha = \alpha^1{}^\frown\alpha^2{}^\frown...{}^\frown\alpha^n, \alpha' = \alpha'^1{}^\frown\alpha'^2{}^\frown...{}^\frown\alpha'^n$ and $\forall i \in [1,n]$, $\alpha^i \sim \alpha'^i$

We note $Forward^e_{(\underline{\mathcal{A}},\underline{g},\underline{Adv})}(\alpha)$ the (clearly) unique α' s. t. $\alpha \sim \alpha'$.

Now we recursively define $\sigma' = Forward^s_{(\underline{\mathcal{A}},\underline{g},\underline{Adv})}(\sigma)$ as follows: Let $(\alpha, \alpha') \in frags^*(\mathcal{E}||g_k(\mathcal{A}_k)||Adv_k) \times frags^*(\mathcal{E}||H_k||Adv_k)$, σ' mimics σ, i. e.

- if $\alpha \sim \alpha'$, $\forall b \in \widehat{sig}(\mathcal{E}||H_k||Adv_k)(lstate(\alpha'))$,
  - if $b \in E^+(lstate(\alpha'))$, $\sigma'(\alpha')(b) = \sigma(\alpha)(b)$
  - if $b \in G^+(lstate(\alpha'))$, $\sigma'(\alpha')(b) = 0$
  - if $b \in F^+(lstate(\alpha'))$, $\sigma'(\alpha')(b) = \sigma(\alpha)(origin(b))$
- if $\alpha' = \alpha''{}^\frown bq'$ with $\alpha \sim \alpha''$ and $b \in F^+(lstate(\alpha''))$, then $\sigma'(\alpha') = \delta_{forward(b)}$.

By construction, for every α' s. t. $|\alpha'| > q_2(k)$, $\sigma'(\alpha') = 0$.

The construction ensures $\epsilon_\sigma(\alpha) = \epsilon_{\sigma'}(\alpha')$ for $\alpha \sim \alpha'$ and $\epsilon_{\sigma'}(\alpha') = 0$ if there is no α'' with α' as prefix with $|\alpha''| = |\alpha'| + 1$ s. t. it exists an execution α verifying $\alpha \sim \alpha''$. Since, by bravery property, for every pair (α, α') with $\alpha \sim \alpha'$, we have $f_{(\mathcal{E}_k, H_k||Adv_k)}(\alpha) = f_{(\mathcal{E}_k, g_k(\mathcal{A}_k)||Adv_k)}(\alpha')$, we obtain $f\text{-}dist_{(\mathcal{E}_k, g_k(\mathcal{A}_k)||Adv_k)}(\sigma) = f\text{-}dist_{(\mathcal{E}_k, H_k||Adv_k)}(\sigma')$.

□

Theorem D.2 (Composability of dynamic secure-emulation). *Let (Sch, f) be a brave pair made of a scheduler schema Sch and an insight function f. Let $b \in \mathbb{N}$. Let $\underline{\mathcal{A}}^1, \underline{\mathcal{A}}^2, ..., \underline{\mathcal{A}}^b$ and $\underline{\mathcal{B}}^1, \underline{\mathcal{B}}^2, ..., \underline{\mathcal{B}}^b$ be pair- wise partially-compatible polynomial-time-bounded structured PSIOA (resp. PCA) families, with $\underline{\mathcal{A}}^i \leq^{Sch,f}_{SE} \underline{\mathcal{B}}^i$ for every $i \in [1,b]$. Then, we have $\widehat{\underline{\mathcal{A}}} \leq_{SE} \widehat{\underline{\mathcal{B}}}$ with $\widehat{\underline{\mathcal{A}}} = \underline{\mathcal{A}}^1||\underline{\mathcal{A}}^2||...||\underline{\mathcal{A}}^b$ and $\widehat{\underline{\mathcal{B}}} = \underline{\mathcal{B}}^1||\underline{\mathcal{B}}^2||...||\underline{\mathcal{B}}^b$.*

Proof. Let $\underline{Adv}$ be an adversary family for $\widehat{\underline{\mathcal{A}}}$ with polynomially bounded description. We need to construct an adversary family $\underline{Sim}$ for $\widehat{\underline{\mathcal{B}}}$ with polynomially bounded description such that: $hide(\widehat{\underline{\mathcal{A}}}||\underline{Adv}, AAct_{\widehat{\underline{\mathcal{A}}}}) \leq^{Sch,f}_{neg,pt} hide(\widehat{\underline{\mathcal{B}}}||\underline{Sim}, AAct_{\widehat{\underline{\mathcal{B}}}})$.

For every $(i,k) \in [1,b] \times \mathbb{N}$. We note g^i_k an arbitrary bijection from $AAct_{\mathcal{A}^i_k}$ to a set of fresh action names, i. e. $\underline{g}^i = \{g^i_k\}_{k\in\mathbb{N}}$ is a renaming of adversary actions for $\underline{\mathcal{A}}^i$. These functions induce a renaming for $\widehat{\underline{\mathcal{A}}}$: $\underline{g} = \{g_k\}_{k\in\mathbb{N}}$ with $\forall k \in \mathbb{N}, g_k = g^1_k \cup ... \cup g^b_k$, i. e. $\forall q \in states(\widehat{\underline{\mathcal{A}}}_k)$, $\forall a \in AAct_{\widehat{\underline{\mathcal{A}}}}(q)$, $g_k(a) = g^i_k(a)$ iff $a \in AAct_{\mathcal{A}^i_k}(q \upharpoonright \mathcal{A}^i_k)$. We recall that compatibility definition for structured PSIOA requires that shared actions of two automata cannot be a adversary actions of their composition. We note $\widehat{\underline{Dum}} = \underline{D}^1||...||\underline{D}^b$, where $\forall i \in [1,b]$, $\underline{D}^i = Dummy(\underline{\mathcal{A}}^i, \underline{g}^i)$. Let $i \in [1,b]$. Since $\underline{\mathcal{A}}^i \leq_{SE} \underline{\mathcal{B}}^i$, then $\exists \underline{DSim}^i$ s. t.

$hide(\underline{\mathcal{A}}^i||\underline{D}^i, AAct^i_{\underline{\mathcal{A}}}) \leq_{neg,pt} hide(\underline{\mathcal{B}}^i||\underline{DSim}^i, AAct^i_{\underline{\mathcal{B}}})$.

We note $\widehat{\underline{DSim}} = \underline{DSim}^1||...||\underline{DSim}^b$.

We observe the following:

$hide(\widehat{\underline{\mathcal{A}}}||\underline{Adv}, AAct_{\widehat{\underline{\mathcal{A}}}}) \equiv^{Sch,f}_{neg,pt}$

$hide(\underline{g}(\widehat{\underline{\mathcal{A}}})||\underline{g}(\underline{Adv}), \underline{g}(AAct_{\widehat{\underline{\mathcal{A}}}})) \leq^{Sch,f}_{neg,pt}$

$hide(\widehat{\underline{\mathcal{A}}}||\widehat{\underline{Dum}}||\underline{g}(\underline{Adv}), \underline{g}(AAct_{\widehat{\underline{\mathcal{A}}}}) \cup AAct_{\widehat{\underline{\mathcal{A}}}}) \leq^{Sch,f}_{neg,pt}$

$hide(\widehat{\underline{\mathcal{B}}}||\widehat{\underline{DSim}}||\underline{g}(\underline{Adv}), \underline{g}(AAct_{\widehat{\underline{\mathcal{A}}}}) \cup AAct_{\widehat{\underline{\mathcal{B}}}}) \equiv^{Sch,f}_{neg,pt}$

$hide(\widehat{\underline{\mathcal{B}}}||hide(\widehat{\underline{DSim}}||\underline{g}(\underline{Adv}), \underline{g}(AAct_{\widehat{\underline{\mathcal{A}}}})), AAct_{\widehat{\underline{\mathcal{B}}}})$

We define $\underline{Sim} = hide(\widehat{\underline{DSim}}||\underline{g}(\underline{Adv}), \underline{g}(AAct_{\widehat{\underline{\mathcal{A}}}}))$

Hence we have shown that for every adversary family $\underline{Adv}$ for $\widehat{\underline{\mathcal{A}}}$ with polynomially bounded description it exists a polynomially bounded adversary $\underline{Sim}$ for $\widehat{\underline{\mathcal{B}}}$ such that: $hide(\widehat{\underline{\mathcal{A}}}||\underline{Adv}, AAct_{\widehat{\underline{\mathcal{A}}}}) \leq^{Sch,f}_{neg,pt} hide(\widehat{\underline{\mathcal{B}}}||\underline{Sim}, AAct_{\widehat{\underline{\mathcal{B}}}})$, which ends the proof.

□