Modeling Bounded Computation in Long-Lived Systems

Ran Canetti, IBM Research
Ling Cheung, Massachusetts Institute of Technology
Dilsun Kaynar, Carnegie Mellon University
Nancy Lynch, Massachusetts Institute of Technology
Olivier Pereira, Université catholique de Louvain

February 28, 2008

Abstract

For many cryptographic protocols, security relies on the assumption that adversarial entities have limited computational power. This type of security degrades progressively over the lifetime of the protocol. However, some cryptographic services (e.g., time-stamping services or digital archives) are long-lived in nature; they are expected to be secure and operational for a very long time (i.e., super-polynomial). In such cases, security cannot be guaranteed in the traditional sense: even information theoretically secure protocols would fail if the attacker has sufficient run time to mount a brute-force attack.

This work proposes a new paradigm for the analysis of long-lived security protocols. We allow entities to be active for a potentially unbounded amount of real time, provided they perform only a polynomial amount of work per unit real time. Moreover, the space used by these entities is allocated dynamically and must be polynomially bounded. We propose a key notion of long-term implementation, which is an adaptation of computational indistinguishability to the long-lived setting. We show that long-term implementation is preserved under polynomial parallel composition and exponential sequential composition. To illustrate the use of this new paradigm, we analyze the long-lived timestamping protocol of Haber and Kamat.

1 Introduction

Nearly all the systems defined and analyzed in cryptographic protocol research are *short-lived*. In these systems, protocol parties can execute only a bounded number of steps, after which the protocol concludes. Depending on the particular model, adversarial entities may perform certain pre- and/or post-computations. While the adversary may be unbounded in these additional phases, it must be bounded during protocol execution. It is typical that security degrades substantially (namely, polynomially) in the length of protocol execution.

In this paper, we turn our attention to the security of *long-lived* cryptographic services. In a long-lived system, protocol parties may be active for an unbounded amount of real time, subject to the condition that only a polynomial amount of work can be done per unit real time. Here the adversary's interaction with the system becomes unbounded, and the adversary may perform an unbounded number of computation steps during protocol execution. This renders traditional security notions insufficient: even information theoretically secure protocols would fail if the adversary has unbounded run time.

Despite the seeming impossibility to defeat a long-lived adversary, there exist long-lived protocols that aim to provide some meaningful form of security [1, 5]. Thus, we set out to formulate a new notion of security that captures the intuitions behind such protocols.

As it turns out, the modeling of long-lived systems requires some non-trivial departures from standard cryptographic modeling. First and foremost, unbounded entities cannot be modeled as *probabilistic polynomial time (PPT)* Turing machines. In search of a suitable alternative, we see the need to distinguish between two types of unbounded computation: steps performed steadily over a long period of time, versus those performed rapidly in a short amount of time. The former conforms with our understanding of boundedness, while the latter clearly does not. Guided by this intuition, we introduce real time explicitly in the Task-PIOA model [3] and impose computational restrictions in terms of *rates*, i.e., number of computation steps per unit real time.

Another interesting challenge is the restriction on space, which traditionally is a non-issue because PPT Turing machines can only access a polynomially bounded amount of space. In the long-lived setting, space restriction warrants new considerations. For instance, we would like to model dynamic allocation of space, as new entities are invoked and old entities die off. This is achieved using the designation of variables. In particular, the state of every entity is represented by a valuation of its variables, and all variables of a dormant entity (either not yet invoked or already killed off) must be set to a special value $\perp$. A system is regarded as bounded only if, at any point in its execution, only a bounded amount of space is needed to maintain all variables with non-$\perp$ values. For example, a sequential composition (in the temporal sense) of an unbounded number of entities is bounded if each entity uses a bounded amount of space.

Having appropriate restrictions on space and computation rates, we define a long-term implementation relation for long-lived systems. This extends the familiar notion of *computational indistinguishability*, where two systems (*real* and *ideal*) are deemed equivalent if their behaviors are indistinguishable from the point of view of a computationally bounded environment. Notice that, in the long-lived setting, an environment with super-polynomial run time can typically distinguish the two systems trivially, e.g., by launching brute force attacks. This holds even if the environment has bounded computation rate. Therefore, our definition does not rule out significant degradation of security in the overall lifetime of a system. Instead, we require that the *rate* of degradation is small at any point in time; in other words, the probability of a new successful attack remains bounded during the lifetime of the system.

To capture this intuition, we introduce a new type of ideal system; namely, we consider ideal systems with special "failure" steps. Whenever a failure step is taken, an ideal system becomes vulnerable to attacks for a limited time. After that, certain damage control mechanisms go into effect, and the ideal system returns to a "good" state and continues to behave ideally. Our implementation relation requires that the real system approximates this type of self-correcting behavior. More precisely, we quantify over all real time points t and require that the real and ideal systems are computationally indistinguishable in the interval $[t, t+q]$ (where q is polynomial in the security parameter), even if no more failure steps are taken by the ideal system in that interval. Notice that we do allow failure steps before time t. This expresses the idea that, despite any security breaches that may have occurred before time t, the success probability of a *fresh* attack in the interval $[t, t+q]$ is small.

We show that our long-term implementation relation is transitive, and is preserved under the operations polynomial parallel composition and exponential sequential composition. The last result highlights the power of our model: we can formulate and prove properties of an exponential number of entities in a meaningful way.

Example: Digital Timestamping As a proof of concept, we present an analysis of the digital timestamping protocol of Haber et al. [1, 4, 5], which was designed to address the problem of content integrity in long-term digital archives. In a nutshell, a digital timestamping scheme takes as input a document d at a specific time t_0, and produces a certificate c that can be used later to verify the existence of d at time t_0. The security requirement is that timestamp certificates are difficult to forge. Haber et al. note that it is inadvisable to use a single digital signature scheme to generate all timestamp certificates, even if signing keys are refreshed periodically. This is because, over time, any single signature scheme may be weakened due to advances in algorithmic research and/or discovery of vulnerabilities. Haber et al. propose a solution in which timestamps must be renewed periodically by generating a new certificate for the pair $\langle d, c\rangle$ using a new signature scheme. Thus, even if the signature scheme used to generate c is broken in the future, the new certificate c' still provides evidence that d existed at the time t_0 stated in the original certificate c.

We model the protocol of Haber et al. as the composition of a dispatcher component and a sequence of signature services. Each signature service "wakes up" at a certain time and is active for a specified amout of time before becoming dormant again. This can be viewed as a regular update of the signature service, which may entail a simple refresh of the signing key, or the adoption of a new signing algorithm. The dispatcher component accepts various timestamp requests and forwards them to the appropriate signature service. We show that the composition of the dispatcher and the signature services is indistinguishable from an ideal system, consisting of the same dispatcher composed with ideal signature functionalities. Specifically, this guarantees that the probability of a new forgery is small at any given point in time, regardless of any forgeries that may have happened in the past.

Related Work In the past decades, the cryptography and concurrency communities have developed rigorous frameworks for modeling protocols, formulating security properties, and proving correctness (e.g., [7, 2, 11, 3, 6]). These models, however, concentrate on short-lived systems where system lifetime is comparable to the computational complexity of individual entities and to the level of security provided. In [10], Müller-Quade and Unruh study *long-term security* of cryptographic protocols. They consider adversaries that try to derive information from the protocol transcript *after* protocol conclusion. This work differs from ours, in that it does not consider long-lived protocol execution. In particular, the adversary of [10] has polynomially bounded interactions with the protocol parties.

2 Task-PIOAs

We review the basics of the task-PIOA framework [3], which has a partial-information scheduling mechanism based on tasks. A task is a set of related actions (e.g., actions representing the same activity but with different parameters). We view tasks as basic units of events, both for real time scheduling and for imposing computational bounds (cf. Sections 3 and 4).

Notations Given a set S, let $\mathsf{Disc}(S)$ denote the set of discrete probability measures on S. For $s \in S$, let $\delta(s)$ denote the *Dirac* measure on s, i.e., $\delta(s)(s) = 1$.

Let V be a set of variables. Each $v \in V$ is associated with a *(static) type* $\mathsf{type}(v)$, which is the set of all possible values of v. We assume that $\mathsf{type}(v)$ is countable and contains the special symbol $\perp$. A *valuation* s for V is a function mapping every $v \in V$ to a value in $\mathsf{type}(v)$. The set of all valuations for V is denoted $val(V)$. Given $V' \subseteq V$, a valuation s' for V' is sometimes referred to as a *partial valuation* for V. Observe that s' induces a (full) valuation $\iota_V(s')$ for V, by assigning $\perp$ to every $v \notin V'$.

Finally, given any set S, we write $S_\perp := S \cup \{\perp\}$, assuming $\perp \notin S$.

PIOA We define a *probabilistic input/output automaton (PIOA)* to be a tuple $\mathcal{A} = \langle V, S, s^{\mathsf{init}}, I, O, H, \Delta\rangle$, where:

(i) V is a set of *state variables* and $S \subseteq val(V)$ is a set of *states*;
(ii) $s^{\mathsf{init}} \in S$ is the *initial* state;
(iii) I, O and H are countable and pairwise disjoint sets of actions, referred to as *input, output and hidden actions*, respectively;
(iv) $\Delta \subseteq S \times (I \cup O \cup H) \times \mathsf{Disc}(S)$ is a *transition relation*.

The set $Act := I \cup O \cup H$ is the *action alphabet* of $\mathcal{A}$. If $I = \emptyset$, then $\mathcal{A}$ is said to be *closed*. The set of *external* actions of $\mathcal{A}$ is $I \cup O$ and the set of *locally controlled* actions is $O \cup H$. Any sequence of external actions is called a *trace*. We write $s.v$ for the value of variable v in state s. An action a is *enabled* in a state s if $\langle s, a, \mu\rangle \in \Delta$ for some μ. We require that $\mathcal{A}$ satisfies the following conditions.

- **Input Enabling:** For every $s \in S$ and $a \in I$, a is enabled in s.
- **Transition Determinism:** For every $s \in S$ and $a \in Act$, there is at most one $\mu \in \mathsf{Disc}(S)$ with $\langle s, a, \mu\rangle \in \Delta$. We write $\Delta(s, a)$ for such μ, if it exists.

Parallel composition for PIOAs is based on synchronization of shared actions. PIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are said to be *compatible* if $V_i \cap V_j = Act_i \cap H_j = O_i \cap O_j = \emptyset$ whenever $i \neq j$. In that case, we define their *composition* $\mathcal{A}_1 \| \mathcal{A}_2$ to be $\langle V_1 \cup V_2, S_1 \times S_2, \langle s_1^{\mathsf{init}}, s_2^{\mathsf{init}}\rangle, (I_1 \cup I_2) \setminus (O_1 \cup O_2),$
$O_1 \cup O_2, H_1 \cup H_2, \Delta\rangle$, where Δ is the set of triples $\langle\langle s_1, s_2\rangle, a, \mu_1 \times \mu_2\rangle$ satisfying: (i) a is enabled in some s_i, and (ii) for every i, if $a \in Act_i$, then $\langle s_i, a, \mu_i\rangle \in \Delta_i$, otherwise $\mu_i = \delta(s_i)$. It is easy to check that input enabling and transition determinism are preserved under composition. Moreover, the definition of composition can be generalized to any finite number of components.

Task-PIOAs To resolve nondeterminism, we make use of the notion of tasks introduced in [8, 3]. Formally, a *task-PIOA* is a pair $\langle \mathcal{A}, \mathcal{R}\rangle$ where $\mathcal{A}$ is a PIOA and $\mathcal{R}$ is a partition of the locally-controlled actions of $\mathcal{A}$. The equivalence classes in $\mathcal{R}$ are called *tasks*. For notational simplicity, we often omit $\mathcal{R}$ and refer to the task-PIOA $\mathcal{A}$. The following axiom is assumed.

- **Action Determinism:** For every state s and every task T, at most one action $a \in T$ is enabled in s.

Unless otherwise stated, terminologies are inherited from the PIOA setting. For instance, if some $a \in T$ is enabled in a state s, then T is said to be *enabled* in s.

Example 1 (Clock automaton). *Figure 1 describes a simple task-PIOA* $\mathsf{Clock}(\mathbb{T})$*, which has a* $\mathsf{tick}(t)$ *output action for every t in some discrete time domain $\mathbb{T}$. For concreteness, we assume $\mathbb{T} = \mathbb{N}$ and simply write* Clock*. There is a single task* tick*, consisting of all* $\mathsf{tick}(t)$ *actions. These clock ticks are produced in order, for $t = 1, 2, \ldots$. In Section 3, we will define a mechanism that ensures each* $\mathsf{tick}(t)$ *occurs exactly at real time t.*

Operations Given compatible task-PIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$, we define their *composition* to be $\langle \mathcal{A}_1 \| \mathcal{A}_2, \mathcal{R}_1 \cup \mathcal{R}_2\rangle$. Note that $\mathcal{R}_1 \cup \mathcal{R}_2$ is an equivalence relation because compatibility requires disjoint sets of locally controlled actions. Moreover, it is easy to check that action determinism is preserved under composition.

```
Clock($\mathbb{T}$)

Signature
  Input:
      none
  Output:
      tick($t : \mathbb{T}$), $t > 0$

Tasks
  tick = {tick($*$)}

States
  $count \in \mathbb{T}$, initially 0

Transitions
  tick($t$)
  Precondition:
      $count = t - 1$
  Effect:
      $count := t$
```

Figure 1: Task-PIOA Code for Clock($\mathbb{T}$)

We also define a *hiding* operator: given $\mathcal{A} = \langle V, S, s^{\mathsf{init}}, I, O, H, \Delta\rangle$ and $S \subseteq O$, $\mathsf{hide}(\mathcal{A}, S)$ is the task-PIOA given by $\langle V, S, s^{\mathsf{init}}, I, O', H', \Delta\rangle$, where $O' = O \setminus S$ and $H' = H \cup S$. This prevents other PIOAs from synchronizing with $\mathcal{A}$ via actions in S: any PIOA with an action in S in its signature is no longer compatible with $\mathcal{A}$.

Executions and traces A *task schedule* for a closed task-PIOA $\langle\mathcal{A}, \mathcal{R}\rangle$ is a finite or infinite sequence $\rho = T_1, T_2, \ldots$ of tasks in $\mathcal{R}$. This induces a well-defined run of $\mathcal{A}$ as follows.

(i) From the start state s^{init}, we *apply* the first task T_1: due to action- and transition-determinism, T_1 specifies at most one transition from s^{init}; if such a transition exists, it is taken, otherwise nothing happens.

(ii) Repeat with remaining T_i's.

Such a run gives rise to a unique *probabilistic execution*, which is a probability distribution over execution paths in $\mathcal{A}$. For finite τ, let $\mathsf{lstate}(\mathcal{A}, \tau)$ denote the state distribution of $\mathcal{A}$ after executing according to τ. A state s is said to be *reachable* under τ if $\mathsf{lstate}(\mathcal{A}, \tau)(s) > 0$. Moreover, the probabilistic execution induces a unique *trace distribution* $\mathsf{tdist}(\mathcal{A}, \tau)$, which is a probability distribution over the set of traces of $\mathcal{A}$. We refer to [3] for more details on these constructions.

3 Real Time Constraints

Recall that our goal is to model entities with unbounded lifetime but bounded processing rates. A natural approach is to introduce real time, so that computational restrictions can be stated in terms of the number of steps performed per unit real time. However, computationally bounded entities cannot maintain real time information to arbitrary precision. Thus, we follow a two-pronged approach: system components maintain discrete approximations of time in their logical state, while task schedules contain real time information.

A *timed* task schedule τ for a closed task-PIOA $\langle\mathcal{A}, \mathcal{R}\rangle$ is a finite or infinite sequence $\langle T_1, t_1\rangle, \langle T_2, t_2\rangle, \ldots$ such that: $T_i \in \mathcal{R}$ and $t_i \in \mathbb{R}_{\geq 0}$ for every i, and $t_1, t_2, \ldots$ is non-decreasing.

Following [9], we associate lower and upper real time bounds to each task. If l and u are, respectively, the lower bound and upper bound for a task T, then the amount of time between consecutive occurrences of T is at least l and at most u. To limit computational power, we impose a rate bound on the number of occurrences of T within an interval I, based on the length of I. A burst bound is also included for modeling flexibility. Formally, a *bound map* for a task-PIOA $\langle\mathcal{A}, \mathcal{R}\rangle$ is a tuple $\langle\mathsf{rate}, \mathsf{burst}, \mathsf{lb}, \mathsf{ub}\rangle$ such that: (i) $\mathsf{rate}, \mathsf{burst}, \mathsf{lb} : \mathcal{R} \to \mathbb{R}_{\geq 0}$, (ii) $\mathsf{ub} : \mathcal{R} \to \mathbb{R}^{\infty}_{>0}$, and (iii) for all $T \in \mathcal{R}$, $\mathsf{lb}(T) \leq \mathsf{ub}(T)$. To ensure that rate and ub can be satisfied simultaneously, we require $\mathsf{rate}(T) \geq 1/\,\mathsf{ub}(T)$ whenever $\mathsf{rate}(T) \neq 0$ and $\mathsf{ub}(T) \neq \infty$. From this point on, we assume that every task-PIOA is associated with a particular bound map.

Given a timed schedule τ and a task T, let $\mathsf{proj}_T(\tau)$ denote the result of removing all pairs $\langle T_i, t_i\rangle$ with $T_i \neq T$. Let d denote a nonnegative real and let I be an interval of the form $[0, t_I]$ for some $t_I \in \mathbb{R}_{\geq 0}$. We say that τ is *valid* for the interval I (under a bound map $\langle\mathsf{rate}, \mathsf{burst}, \mathsf{lb}, \mathsf{ub}\rangle$) if the following hold for every task T.

(i) If the pair $\langle T, t\rangle$ appears in τ, then $t \in I$.

(ii) If $\mathsf{lb}(T) > 0$, then: (a) if $\langle T, t\rangle$ is the first element of $\mathsf{proj}_T(\tau)$, then $t \geq \mathsf{lb}(T)$; (b) for every interval I' of length $d < \mathsf{lb}(T)$, $\mathsf{proj}_T(\tau)$ contains at most one element $\langle T, t\rangle$ with $t \in I'$.

(iii) If $\mathsf{ub}(T) \neq \infty$, then, for every interval $I' \subseteq I$ of length $d > \mathsf{ub}(T)$, $\mathsf{proj}_T(\tau)$ contains at least one element $\langle T, t\rangle$ with $t \in I'$.

(iv) For any interval I' of length d, $\mathsf{proj}_T(\tau)$ contains at most $\mathsf{rate}(T) \cdot d + \mathsf{burst}(T)$ elements $\langle T, t\rangle$ with $t \in I'$.

Note that every timed schedule τ projects to an untimed schedule ρ by removing all real time information t_i, thereby inducing a trace distribution of $\mathcal{A}$. The set of trace distributions induced by all valid timed schedules for $\mathcal{A}$ and $\langle\mathsf{rate}, \mathsf{burst}, \mathsf{lb}, \mathsf{ub}\rangle$ is denoted $\mathsf{TrDists}(\mathcal{A}, \mathsf{rate}, \mathsf{burst}, \mathsf{lb}, \mathsf{ub})$. Since the bound map is typically fixed, we often omit it and write $\mathsf{TrDists}(\mathcal{A})$.

In a parallel composition $\mathcal{A}_1\|\mathcal{A}_2$, the composite bound map is the union of component bound maps:

$$\langle\mathsf{rate}_1 \cup \mathsf{rate}_2, \mathsf{burst}_1 \cup \mathsf{burst}_2, \mathsf{lb}_1 \cup \mathsf{lb}_2, \mathsf{ub}_1 \cup \mathsf{ub}_2\rangle.$$

This is well defined since the task partition of $\mathcal{A}_1\|\mathcal{A}_2$ is $\mathcal{R}_1 \cup \mathcal{R}_2$.

Example 2 (Bound map for Clock**).** *We use upper and lower bounds to ensure that* Clock*'s internal counter evolves at the same rate as real time. Namely, we set* $\mathsf{lb}(\mathsf{tick}) = \mathsf{ub}(\mathsf{tick}) = 1$. *The rate and burst bounds are also set to* 1. *It is not hard to see that, regardless of the system of automata with which* Clock *is composed, we always obtain the unique sequence* $\langle\mathsf{tick}, 1\rangle, \langle\mathsf{tick}, 2\rangle, \ldots$ *when we project a valid schedule to the task* tick.

4 Complexity Bounds

Intuitively, we envision a large collection of task-PIOAs that runs for an unbounded amount of real time. While the number of task-PIOAs in this collection is large, only a bounded number of them will be active simultaneously at any given point in time. Each task-PIOA has bounded memory and bounded computation rates, therefore the overall collection should also satisfy these conditions.

We propose a notion of step bounds that captures these intuitions. Roughly speaking, step bounds limit the amount of computation involved in executing a single action, as well as the amount of space that is allocated as a result of that action. Combining the step bound with the rate and burst bounds of Section 3, we obtain a notion of bounded space and bounded computation rates.

Step Bound We assume some standard bit string encoding for Turing machines and for the names of variables, actions, and tasks. We also assume that variable valuations are encoded in the obvious way, as a list of name/value pairs. Let $\mathcal{A}$ be a task-PIOA with variable set V. Given state s, let $\hat{s}$ denote the partial valuation obtained from s by removing all pairs of the form $\langle v, \bot\rangle$. We have $\iota_V(\hat{s}) = s$, therefore no information is lost by reducing s to $\hat{s}$. This key observation allows us to represent a "large" valuation s with a "condensed" partial valuation $\hat{s}$.

Let $p \in \mathbb{N}$ be given. We say that a state s is p-bounded if the encoding of $\hat{s}$ is at most p bits long. The task-PIOA $\mathcal{A}$ is said to have *step bound* p if the following hold.

(i) For every variable $v \in V$, $\mathsf{type}(v) \subseteq \{0, 1\}^p$.

(ii) The name of every action, task, and variable of $\mathcal{A}$ has length at most p.

(iii) The initial state s^{init} is p-bounded.

(iv) There exists a deterministic Turing machine M_{enable} satisfying: for every p-bounded state s, M_{enable} on input $\hat{s}$ outputs the list of tasks enabled in s.

(v) There exists a probabilistic Turing machine $M_{\mathcal{R}}$ satisfying: for every p-bounded state s and task T, $M_{\mathcal{R}}$ on input $\langle\hat{s}, T\rangle$ decides whether T is enabled in s. If so, $M_{\mathcal{R}}$ computes and outputs a new partial valuation $\hat{s}'$, along with the unique $a \in T$ that is enabled in s. The distribution on $\iota_V(\hat{s}')$ coincides with $\Delta(s, a)$.

(vi) There exists a probabilistic Turing machine M_I satisfying: for every p-bounded state s and action a, M_I on input $\langle\hat{s}, a\rangle$ decides whether a is an input action of $\mathcal{A}$. If so, M_I computes a new partial valuation $\hat{s}'$. The distribution on $\iota_V(\hat{s}')$ coincides with $\Delta(s, a)$.

(vii) The encoding of M_{enable} is at most p bits long, and M_{enable} terminates after at most p steps on every input. The same hold for $M_{\mathcal{R}}$ and M_I.

Thus, step bound p limits the size of action names, which often represent protocol messages. It also limits the number of tasks enabled from any p-bounded state (Condition (iv)) and the complexity of individual transitions (Conditions (v) and (vi)). Finally, Condition (vii) requires all of the Turing machines to have description bounded by p.

Lemma 4.1 below guarantees that a task-PIOA with step bound p will never reach a state in which more than p variables have non-$\perp$ values. The proof is a simple inductive argument.

Lemma 4.1. *Let $\mathcal{A}$ be a task-PIOA with step bound p. For every valid timed task schedule τ and every state s reachable under τ, there are at most p variables v such that $s.v \neq \perp$.*

Proof. By the definition of step bounds, we have s^{init} is p-bounded. For a state s' reachable under schedule τ', let s be a state immediately preceding s' in the probabilistic execution induced by τ'. Thus s is reachable under some prefix of τ. If the transition from s to s' is locally controlled, we use the fact that $M_{\mathcal{R}}$ always terminates after at most p steps, therefore every possible output, including $\hat{s}'$, has length at most p. This implies $\hat{s}'$ is a partial valuation on at most p variables. If the transition from s to s' is an input, we follow the same argument with M_I. $\square$

Lemma 4.2 says that, when we compose task-PIOAs in parallel, the complexity of the composite is proportional to the sum of the component complexities. The proof is similar to that of the full version of [3, Lemma 4.2]. We also note that the hiding operator introduced in Section 2 preserves step bounds.

Lemma 4.2. *Suppose $\{\mathcal{A}_i | 1 \leq i \leq b\}$ is a compatible set of task-PIOAs, where each A_i has step bound $p_i \in \mathbb{N}$. The composition $\|_{i=1}^{b} \mathcal{A}_i$ has step bound $c_{\mathsf{comp}} \cdot \sum_{i=1}^{b} p_i$, where c_{comp} is a fixed constant.*

Turing Machine Simulation Given a closed (i.e., no input actions) task-PIOA $\mathcal{A}$ with step bound p, one can define a nondeterministic Turing machine $M_{\mathcal{A}}$ that simulates the execution of $\mathcal{A}$. The amount of work tape needed by $M_{\mathcal{A}}$ is polynomial in p. As a convention, we write s for the current state and s' for the next state after a transition. Recall that $\hat{s}$ denotes the partial valuation obtained from s by removing all pairs of the form $\langle v, \perp \rangle$. $M_{\mathcal{A}}$ maintains this partial valuation on its work tape. The following procedure is repeated indefinitely by $\mathcal{A}$.

(i) From state s, $M_{\mathcal{A}}$ gives to M_{enable} the partial valuation $\hat{s}$ currently stored on the work tape.

(ii) The run stops if M_{enable} outputs nothing. Otherwise, a task T is chosen nondeterministically from the output of M_{enable} and $\langle \hat{s}, T \rangle$ is given to $M_{\mathcal{R}}$.

(iii) $M_{\mathcal{R}}$ returns $\langle \hat{s}', a \rangle$. $M_{\mathcal{A}}$ checks every variable v appearing in $\hat{s}'$: if v appears in $\hat{s}$, then the value of v is updated on the work tape; otherwise, $M_{\mathcal{A}}$ allocates enough space to store the name of v and the value $\hat{s}'(v)$. Finally, $M_{\mathcal{A}}$ checks for variables appearing in $\hat{s}$ but not $\hat{s}'$. The storage for those variables is freed.

Since the name and type of every variable are also bounded by p, we can infer from Lemma 4.1 that the space needed to represent a reachable state is polynomial in p (in fact, on the order of p^2). Moreover, the amount of work tape needed by M_{enable} and $M_{\mathcal{R}}$ is on the order of p, because these Turing machines execute at most p steps at each activation.[1] Therefore, the total amount of work tape needed by $M_{\mathcal{A}}$ is polynomial in p.

Overall Bound We now put together real time bounds and step bounds. To do so, we represent global time using the clock automaton Clock (Figure 1). Let $p \in \mathbb{N}$ be given and let $\mathcal{A}$ be a task-PIOA compatible with Clock. We say that $\mathcal{A}$ is *p-bounded* if the following hold.

(i) $\mathcal{A}$ has step bound p.

(ii) For every task T of $\mathcal{A}$, $\mathsf{rate}(T)$ and $\mathsf{burst}(T)$ are both at most p.

(iii) For every $t \in \mathbb{N}$, let S_t denote the set of states s of $\mathcal{A} \| \mathsf{Clock}$ such that s is reachable under some valid schedule τ and $s.count = t$. There are at most p tasks T such that T is enabled in some $s \in S_t$.

Conditions (i) and (ii) are self-explanatory. Condition (iii) ensures that the enabling of tasks does not change too rapidly. Without this restriction, $\mathcal{A}$ would be able to cycle through a large number of tasks between two clock ticks, without violating the rate bound of any individual task.

Task-PIOA Families We now extend our definitions to task-PIOA families, indexed by a *security parameter* k. More precisely, a *task-PIOA family* $\bar{\mathcal{A}}$ is an indexed set $\{\mathcal{A}_k\}_{k \in \mathbb{N}}$ of task-PIOAs. Given $p : \mathbb{N} \to \mathbb{N}$, we say that $\bar{\mathcal{A}}$ is *p-bounded* just in case: for all k, $\mathcal{A}_k$ is $p(k)$-bounded. If p is a polynomial, then we say that $\bar{\mathcal{A}}$ is *polynomially bounded*. The notions of compatibility and parallel composition for task-PIOA families are defined pointwise.

We remark that our notion of closed (i.e., no input actions), polynomially bounded families is reminiscent of the traditional notion of PSPACE (which is equivalent to nondeterministic PSPACE). Our setting is slightly richer, because we also talk about rates of computation with respect to real time. Thus, we can distinguish machines that compute in large bursts from those that compute at a steady rate.

[1]Note that we are not concerned with M_I here, because $\mathcal{A}$ is closed.

Example: Signature Service We now present an example of a polynomially bounded family of task-PIOAs.

A *signature scheme* Sig consists of three algorithms: KeyGen, Sign and Verify. KeyGen is a probabilistic algorithm that outputs a signing-verification key pair $\langle sk, vk\rangle$. Sign is a probabilistic algorithm that produces a signature σ from a message m and the key sk. Finally, Verify is a deterministic algorithm that maps $\langle m, \sigma, vk\rangle$ to a boolean. The signature σ is said to be *valid* for m and vk if $\mathsf{Verify}(m, \sigma, vk) = 1$.

Let SID be a domain of service identifiers. For each $j \in SID$, we build a signature service as a family of task-PIOAs indexed by security parameter k. Specifically, we define three task-PIOAs, $\mathsf{KeyGen}(k, j)$, $\mathsf{Signer}(k, j)$, and $\mathsf{Verifier}(k, j)$ for every pair $\langle k, j\rangle$. We assume a function $\mathsf{alive} : \mathbb{T} \to 2^{SID}$ such that, for every t, $\mathsf{alive}(t)$ is the set of services alive at time t. The lifetime of each service j is then given by $\mathsf{aliveTimes}(j) := \{t \in \mathbb{T} | j \in \mathsf{alive}(t)\}$, which is assumed to be a finite set of consecutive numbers.

For every security parameter k, we assume the following finite domains: RID_k (request identifiers), M_k (messages to be signed) and Σ_k (signatures). The representations of elements in these domains are bounded by $p(k)$, for some polynomial p. Similarly, the domain $\mathbb{T}_k$ consists of natural numbers representable using $p(k)$ bits. When combined with the automaton Clock (Figure 1), the input $\mathsf{tick}(t)$ actions allow the components to record discrete time information in the state variable $clock$.

KeyGen $\mathsf{KeyGen}(k, j)$ chooses a signing key $mySK$ and a corresponding verification key $myVK$. This is done exactly once, at any time when service j is alive. The two keys are output separately, via actions $\mathsf{signKey}(sk)_j$ and $\mathsf{verKey}(vk)_j$. The signing key goes to $\mathsf{Signer}(k, j)$, while the verification key may go to several other components.

The code for $\mathsf{KeyGen}(k, j)$ is given in Figure 2. As we mentioned before, the $\mathsf{tick}(t)$ action brings in the current time. If j is alive at time t, then $clock$ is set to the current time t. Also, if j has just become alive, as evidenced by the fact that the $awake$ flag is currently $\perp$, the $awake$ flag is set to $true$. On the other hand, if j is no longer alive at time t, all variables are set to $\perp$.

The $\mathsf{chooseKeys}$ action uses KeyGen_j to choose the key pair, and is enabled only when j is awake and the keys are currently $\perp$. Note that the KeyGen algorithm is indexed by j, because different services may use different algorithms. The same applies to Sign_j in $\mathsf{Signer}(k, j)$ and Verify_j in $\mathsf{Verifier}(k, j)$. The $\mathsf{signKey}$ and verKey actions output the keys, and they are enabled only when j is awake and the keys have been chosen.

Signer $\mathsf{Signer}(k, j)$ receives the signing key from another component, e.g., $\mathsf{KeyGen}(k, j)$. It then responds to signing requests by running the Sign_j algorithm on the given message m and the received signing key sk. Figure 3 presents the code for $\mathsf{Signer}(k, j)$, which is fairly self-explanatory.

The data type que_k represents queues with maximum length $p(k)$, where p is a polynomial. The enqueue operation automatically discards the new entry if the queue is already of length $p(k)$. This models the fact that $\mathsf{Signer}(k, j)$ has a bounded amount of memory. For concreteness, we assume that p is the constant function $\underline{1}$ for the queues $toSign$ and $signed$.

$\mathsf{Verifier}$ $\mathsf{Verifier}(k, j)$ accepts verification requests and simply runs the Verify_j algorithm. The code appears in Figure 4. Again, all queues have maximum length 1.

Assuming the algorithms KeyGen_j, Sign_j and Verify_j are polynomial time, it not hard to check that the composite $\mathsf{KeyGen}(k, j) \| \mathsf{Signer}(k, j) \| \mathsf{Verifier}(k, j)$ has step bound $p(k)$ for some polynomial p. If $\mathsf{rate}(T)$ and $\mathsf{burst}(T)$ are at most $p(k)$ for every T, then the composite is $p(k)$-bounded. The family $\{\mathsf{KeyGen}(k, j) \| \mathsf{Signer}(k, j) \| \mathsf{Verifier}(k, j)\}_{k\in\mathbb{N}}$ is therefore polynomially bounded.

5 Long-Term Implementation Relation

Much of modern cryptography is based on the notion of computational indistinguishability. For instance, an encryption algorithm is (chosen-plaintext) secure if the ciphertexts of two distinct but equal-length messages are indistinguishable from each other, even if the plaintexts are generated by the distinguisher itself. The key assumption is that the distinguisher is computationally bounded, so that it cannot launch a brute force attack. In this section, we adapt this notion of indistinguishability to the long-lived setting.

We define an implementation relation based on closing environments and acceptance probabilities. Let $\mathcal{A}$ be a closed task-PIOA with output action acc and task $\{\mathsf{acc}\}$. Let τ be a timed task schedule for $\mathcal{A}$. The *acceptance probability* of $\mathcal{A}$ under τ is: $\mathbf{P}_{\mathsf{acc}}(\mathcal{A}, \tau) := \Pr[\beta \text{ contains } \mathsf{acc} : \beta \leftarrow_{\mathsf{R}} \mathsf{tdist}(\mathcal{A}, \tau)]$; that is, the probability that a trace

$\mathsf{KeyGen}(k : \mathbb{N}, j : SID)$

```
Signature

Input:
    $\mathsf{tick}(t : \mathbb{T}_k)$
Output:
    $\mathsf{signKey}(sk : 2^k)_j$
    $\mathsf{verKey}(vk : 2^k)_j$
Internal:
    $\mathsf{chooseKeys}_j$
```

```
Tasks
$\mathsf{verKey}_j = \{\mathsf{verKey}(*)_j\}$
$\mathsf{signKey}_j = \{\mathsf{signKey}(*)_j\}$
$\mathsf{chooseKeys}_j = \{\mathsf{chooseKeys}_j\}$

States
$awake : \{\mathsf{true}\}_\perp$, init $\perp$
$clock : (\mathbb{T}_k)_\perp$, init $\perp$
$mySK : (2^k)_\perp$, init $\perp$
$myVK : (2^k)_\perp$, init $\perp$
```

```
Transitions
$\mathsf{tick}(t)$
Effect:
    if $j \in \mathsf{alive}(t)$ then
        $clock := t$
        if $awake = \perp$ then
            $awake := true$
    else
        $awake, clock, mySK,$
            $myVK := \perp$

$\mathsf{chooseKeys}_j$
Precondition:
    $awake = true$
    $mySK = myVK = \perp$
Effect:
    $\langle mySK, myVK \rangle$
        $\leftarrow \mathsf{KeyGen}_j(1^k)$
```

```
$\mathsf{signKey}(sk)_j$
Precondition:
    $awake = true$
    $sk = mySK \neq \perp$
Effect:
    none

$\mathsf{verKey}(vk)_j$
Precondition:
    $awake = true$
    $vk = myVK \neq \perp$
Effect:
    none
```

Figure 2: Task-PIOA Code for $\mathsf{KeyGen}(k, j)$

drawn from the distribution $\mathsf{tdist}(\mathcal{A}, \tau)$ contains the action acc. If $\mathcal{A}$ is not necessarily closed, we include a closing environment. A task-PIOA Env is an *environment* for $\mathcal{A}$ if it is compatible with $\mathcal{A}$ and $\mathcal{A}\|\mathsf{Env}$ is closed. From here on, we assume that every environment has output action acc.

In the short-lived setting, we say that a system $\mathcal{A}_1$ implements another system $\mathcal{A}_2$ if every run of $\mathcal{A}_1$ can be "matched" by a run of $\mathcal{A}_2$ such that no probabilistic polynomial time environment can distinguish the two runs. As we discussed in the introduction, this type of definition is too strong for the long-lived setting, because we must allow environments with unbounded total run time (as long as they have bounded rate and space).

For example, consider the timestamping protocol of [5, 4] described in Section 1. After running for a long period of real time, a distinguisher environment may be able to forge signatures from a much earlier time period. As a result, it can distinguish the real system from the ideal one in the traditional sense. However, the essence of the protocol is that such failures can in fact be tolerated, because the environment cannot forge *recent* signatures, after a new, uncompromised signature service becomes active.

This timestamping example suggests that we need a new notion of indistinguishability that "ignores" a large part of the execution history. That is, even when an attacker succeeds in breaking old cryptographic services, the system remains secure as long as the attacker cannot break the services used in recent history. Our new implementation relation aims to capture this intuition.

First we state a comparability condition on task-PIOA: $\mathcal{A}_1$ and $\mathcal{A}_2$ are said to be *comparable* if they have the same external interface, that is, $I_1 = I_2$ and $O_1 = O_2$. In this case, every environment E for $\mathcal{A}_1$ is also an environment for $\mathcal{A}_2$, provided E is compatible with $\mathcal{A}_2$.

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be comparable task-PIOAs, and let $\mathbf{F}$ be a set of tasks of $\mathcal{A}_2$. Given $t \in \mathbb{R}_{\geq 0}$ and an environment Env for both $\mathcal{A}_1$ and $\mathcal{A}_2$, we set up two experiments. In the first experiment, Env interacts with $\mathcal{A}_1$ according to some valid task schedule τ_1 of $\mathcal{A}_1\|\mathsf{Env}$. In the second experiment, Env interacts with $\mathcal{A}_2$ according to some valid task schedule τ_2 of $\mathcal{A}_2\|\mathsf{Env}$, where τ_2 does not contain any tasks from F from time t onwards. Intuitively, the tasks in F corresponds to certain protocol vulnerabilities. Before time t, $\mathcal{A}_2$ may be vulnerable to certain attacks, matching any failures that may have occurred in $\mathcal{A}_1$. (An example of a failure is a forgery in the case of signatures.) At time t and

$\mathsf{Signer}(k : \mathbb{N}, j : SID)$

```
Signature

Input:
    $\mathsf{tick}(t : \mathbb{T}_k)$
    $\mathsf{signKey}(sk : 2^k)_j$
    $\mathsf{reqSign}(rid : RID_k,$
        $m : M_k)_j$
Output:
    $\mathsf{respSign}(rid : RID_k,$
        $\sigma : \Sigma_k)_j$
Internal:
    $\mathsf{sign}(rid : RID_k, m : M_k)_j$

Tasks
$\mathsf{respSign}_j = \{\mathsf{respSign}(*, *)_j\}$
$\mathsf{sign}_j = \{\mathsf{sign}(*, *)_j$

States
$awake : \{\mathsf{true}\}_\perp$, init $\perp$
$clock : (\mathbb{T}_k)_\perp$, init $\perp$
$mySK : (2^k)_\perp$, init $\perp$
$toSign : \mathsf{que}(RID_k \times M_k)_\perp$,
init $\perp$
$signed : \mathsf{que}(RID_k \times \Sigma_k)_\perp$,
init $\perp$

Transitions
$\mathsf{tick}(t)$
Effect:
    if $j \in \mathsf{alive}(t)$ then
        $clock := t$
        if $awake = \perp$ then
            $awake := true$
            $toSign, signed$
                $:=$ empty
    else
        $awake, clock, mySK,$
        $toSign, signed := \perp$

$\mathsf{signKey}(sk)_j$
Effect:
    if $awake = true$
        $\wedge mySK = \perp$
    then $mySK := sk$

$\mathsf{reqSign}(rid, m)_j$
Effect:
    if $awake = true$
        $\wedge \neg\, \mathsf{full}(toSign)$
    then $toSign :=$
        $\mathsf{enq}(toSign, \langle rid, m \rangle)$

$\mathsf{sign}(rid, m)_j$
local $\sigma : \Sigma$
Precondition:
    $awake = true$
    $\mathsf{head}(toSign) = \langle rid, m \rangle$
    $mySK \neq \perp$
Effect:
    $toSign := \mathsf{deq}(toSign)$
    $\sigma \leftarrow \mathsf{Sign}_j(m, mySK)$
    $signed :=$
        $\mathsf{enq}(signed, \langle rid, \sigma \rangle)$

$\mathsf{respSign}(rid, \sigma)_j$
Precondition:
    $awake = true$
    $head(signed) = \langle rid, \sigma \rangle$
Effect:
    $signed := \mathsf{deq}(signed)$
```

Figure 3: Task-PIOA Code for $\mathsf{Signer}(k, j)$

afterwards, $\mathcal{A}_2$ closes the vulnerabilities.

We require that, for any valid τ_1, there exists a valid τ_2 as above such that the two executions are identical up to time t from the point of view of the environment. That is, the acceptance probabilities in these experiments are the same up to time t and Env has the same state distribution immediately before time t. Moreover, the two executions are overall *computationally indistinguishable*, namely, the difference in acceptance probabilities in these two experiments is negligible as long as Env is computationally bounded.

Given a task schedule $\tau = \langle T_1, t_1 \rangle, \langle T_2, t_2 \rangle, \ldots$, let $\mathsf{trunc}_{\geq t}(\tau)$ denote the result of removing all pairs $\langle T_i, t_i \rangle$ with $t_i \geq t$. If τ is a schedule of $\mathcal{A} \| \mathcal{B}$, then we define $\mathsf{proj}_{\mathcal{B}}(\tau)$ to be the result of removing all $\langle T_i, t_i \rangle$ where T_i is *not* a task of $\mathcal{B}$. Moreover, let $\mathsf{lstate}_{\mathcal{B}}(\mathcal{A} \| \mathcal{B}, \tau)$ denote the end state distribution of $\mathcal{B}$ after executing with $\mathcal{A}$ under the schedule τ (assuming τ is finite).

Definition 5.1. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be comparable task-PIOAs that are both compatible with* Clock. *Let F be a set of tasks of $\mathcal{A}_2$, and let $p, q \in \mathbb{N}$ and $\epsilon \in \mathbb{R}_{\geq 0}$ be given. We say that $\mathcal{A}_1 \leq^F_{p,q,\epsilon} \mathcal{A}_2$ if: for every $t \in \mathbb{R}_{\geq 0}$, every environment* Env *of the form* Env$'\|$Clock *with* Env$'$ *being p-bounded, and every valid timed schedule τ_1 for $\mathcal{A}_1 \|$*Env *for the interval $[0, t+q]$, there exists valid timed schedule τ_2 for $\mathcal{A}_2 \|$*Env *for the interval $[0, t+q]$ such that:*

(i) $\mathbf{P}_{\mathsf{acc}}(\mathcal{A}_1 \| \mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_1)) = \mathbf{P}_{\mathsf{acc}}(\mathcal{A}_2 \| \mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_2))$;
(ii) $\mathsf{lstate}_{\mathsf{Env}}(\mathcal{A}_1 \| \mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_1)) = \mathsf{lstate}_{\mathsf{Env}}(\mathcal{A}_2 \| \mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_2))$;
(iii) $\mathsf{proj}_{\mathsf{Env}}(\tau_1) = \mathsf{proj}_{\mathsf{Env}}(\tau_2)$;
(iv) *τ_2 does not contain any pairs of the form $\langle T_i, t_i \rangle$ where $T_i \in F$ and $t_i \geq t$;*

Verifier$(k : \mathbb{N}, j : SID)$

```
Signature                                   Tasks
                                            respVer_j = {respVer(*,*)_j}
Input:                                      verify_j = {verify(*,*,*)_j}
    tick(t : T_k)
    verKey(vk : 2^k)_j                      States
    reqVer(rid : RID_k,                     awake : {true}_⊥, init ⊥
        m : M_k, σ : Σ_k)_j                 clock : (T_k)_⊥, init ⊥
Output:                                     myVK : (2^k)_⊥, init ⊥
    respVer(rid : RID_k,                    toVer : que(RID_k × M_k
        b : Bool)_j                         ×Σ_k)_⊥, init ⊥
Internal:                                   verified : que(RID_k × M_k
    verify(rid : RID_k,                     ×Σ_k)_⊥, init ⊥
        m : M_k, σ : Σ_k)_j

Transitions
 tick(t)                                     verify(rid, m, σ)_j
 Effect:                                     local b : Bool
     if j ∈ alive(t) then                    Precondition:
        clock := t                               awake = true
        if awake = ⊥ then                           ∧myVK ≠ ⊥
           awake := true                         head(toVer) = ⟨rid, m, σ⟩
           toVer, verified                   Effect:
              := empty                           toVer := deq(toVer)
     else                                        b := Verify_j(m, σ, myVK)
        awake, clock, myVK,                      verified :=
        toVer, verified := ⊥                          enq(verified, ⟨rid, b⟩)

 verKey(vk)_j                                respVer(rid, b)_j
 Effect:                                     Precondition:
     if awake = true                             awake = true
        ∧myVK = ⊥                                head(verified) = ⟨rid, b⟩
     then myVK := vk                         Effect:
                                                 verified := deq(verified)
 reqVer(rid, m, σ)_j
 Effect:
     if awake = true
        ∧¬ full(toVer)
     then toVer :=
           enq(toVer, ⟨rid, m, σ⟩)
```

Figure 4: Task-PIOA Code for Verifier(k, j)

(v) $|\mathbf{P}_{\mathsf{acc}}(\mathcal{A}_1\|\mathsf{Env}, \tau_1) - \mathbf{P}_{\mathsf{acc}}(\mathcal{A}_2\|\mathsf{Env}, \tau_2)| \leq \epsilon$.

The following lemma says that $\leq^F_{p,q,\epsilon}$ (Definition 5.1) is transitive up to additive errors.

Lemma 5.2. *Let $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ be comparable task-PIOAs, and let F_2 be a set of tasks of $\mathcal{A}_2$ and F_3 be a set of tasks of $\mathcal{A}_3$. Let $p, q \in \mathbb{N}$ and $\epsilon \in \mathbb{R}_{\geq 0}$ be given. Assume that $\mathcal{A}_1 \leq^{F_2}_{p,q,\epsilon_1} \mathcal{A}_2$ and $\mathcal{A}_2 \leq^{F_3}_{p,q,\epsilon_2} \mathcal{A}_3$. Then $\mathcal{A}_1 \leq^{F_3}_{p,q,\epsilon_1+\epsilon_2} \mathcal{A}_3$.*

Proof. Let $t \in \mathbb{R}_{\geq 0}$, a p-bounded environment Env of the form $\mathsf{Env}'\|\mathsf{Clock}$, and a valid timed schedule τ_1 for $\mathcal{A}_1\|\mathsf{Env}$ for the interval $[0, t+q]$ be given. Choose τ_2 for $\mathcal{A}_2\|\mathsf{Env}$ according to the assumption $\mathcal{A}_1 \leq^{F_2}_{p,q,\epsilon_1} \mathcal{A}_2$. Using τ_2, choose τ_3 for $\mathcal{A}_3\|\mathsf{Env}$ according to the assumption $\mathcal{A}_2 \leq^{F_3}_{p,q,\epsilon_2} \mathcal{A}_3$.

Clearly, we have

- $\mathbf{P}_{\mathsf{acc}}(\mathcal{A}_1\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_1))$
  $= \mathbf{P}_{\mathsf{acc}}(\mathcal{A}_2\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_2))$
  $= \mathbf{P}_{\mathsf{acc}}(\mathcal{A}_3\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_3))$;

- $\mathsf{lstate}_{\mathsf{Env}}(\mathcal{A}_1\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_1))$
  $= \mathsf{lstate}_{\mathsf{Env}}(\mathcal{A}_2\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_2))$
  $= \mathsf{lstate}_{\mathsf{Env}}(\mathcal{A}_3\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_3))$;

- $\mathsf{proj}_{\mathsf{Env}}(\tau_1) = \mathsf{proj}_{\mathsf{Env}}(\tau_2) = \mathsf{proj}_{\mathsf{Env}}(\tau_3)$.

It is also immediate that τ_3 does not contain any pairs of the form $\langle T_i, t_i \rangle$ where $T_i \in F_3$ and $t_i \geq t$. Finally,

$$\begin{aligned} & | \, \mathbf{P}_{\mathsf{acc}}(\mathcal{A}_1 \| \mathsf{Env}, \tau_1) - \mathbf{P}_{\mathsf{acc}}(\mathcal{A}_3 \| \mathsf{Env}, \tau_3) | \\ & \leq | \, \mathbf{P}_{\mathsf{acc}}(\mathcal{A}_1 \| \mathsf{Env}, \tau_1) - \mathbf{P}_{\mathsf{acc}}(\mathcal{A}_2 \| \mathsf{Env}, \tau_2) | \\ & \quad + | \, \mathbf{P}_{\mathsf{acc}}(\mathcal{A}_2 \| \mathsf{Env}, \tau_2) - \mathbf{P}_{\mathsf{acc}}(\mathcal{A}_3 \| \mathsf{Env}, \tau_3) | \\ & \leq \epsilon_1 + \epsilon_2. \end{aligned}$$

□

The relation $\leq^F_{p,q,\epsilon}$ can be extended to task-PIOA families as follows. Let $\bar{\mathcal{A}}_1 = \{(\bar{\mathcal{A}}_1)_k\}_{k \in \mathbb{N}}$ and $\bar{\mathcal{A}}_2 = \{(\bar{\mathcal{A}}_2)_k\}_{k \in \mathbb{N}}$ be pointwise comparable task-PIOA families. Let $\bar{F}$ be a family of sets such that each $(\bar{F})_k$ is a set of tasks of $(\bar{\mathcal{A}}_2)_k$. Let $\epsilon : \mathbb{N} \to \mathbb{R}_{\geq 0}$ and $p, q : \mathbb{N} \to \mathbb{N}$ be given. We say that $\bar{\mathcal{A}}_1 \leq^{\bar{F}}_{p,q,\epsilon} \bar{\mathcal{A}}_2$ just in case $(\bar{\mathcal{A}}_1)_k \leq^{(\bar{F})_k}_{p(k),q(k),\epsilon(k)} (\bar{\mathcal{A}}_2)_k$ for every k.

Restricting our attention to negligible error and polynomial time bounds, we obtain the long-term implementation relation $\leq^{\bar{F}}_{\mathsf{neg,pt}}$. Formally, a function $\epsilon : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is said to be *negligible* if, for every constant $c \in \mathbb{N}$, there exists $k_0 \in \mathbb{N}$ such that $\epsilon(k) < \frac{1}{k^c}$ for all $k \geq k_0$. (That is, ϵ diminishes more quickly than the reciprocal of any polynomial.) Given task-PIOA families $\bar{\mathcal{A}}_1$ and $\bar{\mathcal{A}}_2$ and task set family $\bar{F}$ of $\bar{\mathcal{A}}_2$, we say that $\bar{\mathcal{A}}_1 \leq^{\bar{F}}_{\mathsf{neg,pt}} \bar{\mathcal{A}}_2$ if $\forall p, q \, \exists \epsilon \; \bar{\mathcal{A}}_1 \leq^{\bar{F}}_{p,q,\epsilon} \bar{\mathcal{A}}_2$, where p, q are polynomials and ϵ is a negligible function.

Lemma 5.3 (Transitivity of $\leq^{\bar{F}}_{\mathsf{neg,pt}}$). *Let $\bar{\mathcal{A}}_1$, $\bar{\mathcal{A}}_2$, and $\bar{\mathcal{A}}_3$ be comparable task-PIOA families. Let $\bar{F}_2$ be a task set family of $\bar{\mathcal{A}}_2$ and let $\bar{F}_3$ be a task set family of $\bar{\mathcal{A}}_3$. Suppose $\bar{\mathcal{A}}_1 \leq^{\bar{F}_2}_{\mathsf{neg,pt}} \bar{\mathcal{A}}_2$ and $\bar{\mathcal{A}}_2 \leq^{\bar{F}_3}_{\mathsf{neg,pt}} \bar{\mathcal{A}}_3$. Then $\bar{\mathcal{A}}_1 \leq^{\bar{F}_3}_{\mathsf{neg,pt}} \bar{\mathcal{A}}_3$.*

Proof. Given polynomials p and q, choose negligible functions ϵ_1 and ϵ_2 according to the assumptions. Then $\epsilon_1 + \epsilon_2$ is negligible. By Lemma 5.2, we have $\bar{\mathcal{A}}_1 \leq^{\bar{F}_3}_{p,q,\epsilon_1+\epsilon_2} \bar{\mathcal{A}}_3$. □

6 Ideal Signature Functionality

In this section, we specify an *ideal signature functionality* SigFunc, and show that it is implemented, in the sense of our $\leq^{\bar{F}}_{\mathsf{neg,pt}}$ definition, by the real signature service of Section 4.

As with KeyGen, Signer, and Verifier, each instance of SigFunc is parameterized with a security parameter k and an identifier j. The code for $\mathsf{SigFunc}(k, j)$ appears in Figure 5. It is very similar to the composition of $\mathsf{Signer}(k, j)$ and $\mathsf{Verifier}(k, j)$. The important difference is that $\mathsf{SigFunc}(k, j)$ maintains an additional variable $history$, which records the set of signed messages. In addition, $\mathsf{SigFunc}(k, j)$ has an internal action fail_j, which sets a boolean flag $failed$. If $failed = \mathsf{false}$, then $\mathsf{SigFunc}(k, j)$ uses $history$ to answer verification requests: a signature is rejected if the submitted message is not in $history$, even if Verify_j returns 1. If $failed = \mathsf{true}$, then $\mathsf{SigFunc}(k, j)$ bypasses the check on $history$, so that its answers are identical to those from the real signature service.

Recall that, for every task T of the real signature service, $\mathsf{rate}(T)$ and $\mathsf{burst}(T)$ are bounded by $p(k)$ for some polynomial p. We assume that the same bound applies to $\mathsf{SigFunc}(k, j)$. Since $\mathsf{aliveTimes}(j)$ is a finite set of consecutive numbers, it represents essentially an interval whose length is constant in the security parameter k. Therefore, $p(k)$ gives rise to a bound $p'(k)$ on the maximum number of signatures generated by $\mathsf{SigFunc}(k, j)$, where p' is also polynomial. We set the maximum length of the queue $history$ to $p'(k)$. All other queues have maximum length 1.

We claim that the real signature service implements the ideal signature functionality. The proof relies on a reduction to standard properties of a signature scheme, namely, completeness and existential unforgeability, as defined below.

Definition 6.1. *A signature scheme* $\mathsf{Sig} = \langle \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify} \rangle$ *is* complete *if* $\mathsf{Verify}(m, \sigma, vk) = 1$ *whenever* $\langle sk, vk \rangle \leftarrow \mathsf{KeyGen}(1^k)$ *and* $\sigma \leftarrow \mathsf{Sign}(sk, m)$. *We say that* Sig *is* existentially unforgeable *under adaptive chosen message attacks (or EUF-CMA secure) if no probabilistic polynomial-time forger has non-negligible success probability in the following game.*

Setup *The challenger runs* KeyGen *to obtain* $\langle sk, vk \rangle$ *and gives the forger* vk.

Query *The forger submits message* m. *The challenger responds with signature* $\sigma \leftarrow \mathsf{Sign}(m, sk)$. *This may be repeated adaptively.*

$\mathsf{SigFunc}(k : \mathbb{N}, j : SID)$

```
Signature

Input:
    $I_{\mathsf{Verifier}} \cup I_{\mathsf{Signer}}$
Output:
    $O_{\mathsf{Verifier}} \cup O_{\mathsf{Signer}}$
Internal:
    $H_{\mathsf{Verifier}} \cup H_{\mathsf{Signer}} \cup \{\mathsf{fail}_j\}$

Tasks
$\mathcal{R}_{\mathsf{Signer}} \cup \mathcal{R}_{\mathsf{Verifier}} \cup \{\{\mathsf{fail}_j\}\}$

States
All variables of Signer
and Verifier
$history : \mathsf{que}(M_k)_\perp$, init $\perp$
$failed : \{\mathsf{true}, \mathsf{false}\}_\perp$, init $\perp$

Transitions
Same as Signer and Verifier,
except the following:

$\mathsf{tick}(t)$
Effect:
    if $j \in \mathsf{alive}(t)$ then
        $clock := t$
        if $awake = \perp$ then
            $awake := true$
            $toSign, toVer,$
            $signed, verified$
                := empty
            $history := \emptyset$
            $failed := \mathsf{false}$
    else
        $awake, clock, mySK,$
        $myVK, toSign, toVer,$
        $signed, history, verified,$
        $failed := \perp$

$\mathsf{fail}_j$
Precondition:
    $awake = \mathsf{true}$
Effect:
    $failed := \mathsf{true}$

$\mathsf{sign}(rid, m)_j$
local $\sigma : \Sigma$
Precondition:
    $awake = true$
      $\wedge mySK \neq \perp$
    $\mathsf{head}(toSign) = \langle rid, m \rangle$
Effect:
    $toSign := \mathsf{deq}(toSign)$
    $\sigma := \mathsf{Sign}_j(m, mySK)$
    $signed :=$
        $\mathsf{enq}(signed, \langle rid, \sigma \rangle)$
    $history :=$
        $\mathsf{enq}(history, m)$

$\mathsf{verify}(rid, m, \sigma)_j$
Local $b : Bool$
Precondition:
    $awake = true$
      $\wedge myVK \neq \perp$
    $\mathsf{head}(toVer) = \langle rid, m, \sigma \rangle$
Effect:
    $toVer := \mathsf{deq}(toVer)$
    $b := (\mathsf{Verify}(m, \sigma, myVK)$
      $\wedge (m \in history \vee failed))$
    $verified :=$
        $\mathsf{enq}(verified, \langle rid, b \rangle)$
```

Figure 5: Code for $\mathsf{SigFunc}(k, j)$

Output *The forger outputs a pair $\langle m^*, \sigma^* \rangle$ and he wins if m^* is not among the messages submitted during the query phase and* $\mathsf{Verify}(m^*, \sigma^*, vk) = 1$.

For all $k \in \mathbb{N}$ and $j \in SID$, we define $\mathsf{RealSig}(j)_k$ to be $\mathsf{hide}(\mathsf{KeyGen}(k, j) \| \mathsf{Signer}(k, j) \| \mathsf{Verifier}(k, j), \mathsf{signKey}_j)$ and $\mathsf{IdealSig}(j)_k$ to be $\mathsf{hide}(\mathsf{KeyGen}(k, j) \| \mathsf{SigFunc}(k, j), \mathsf{signKey}_j)$.

These automata are gathered into families in the obvious way: $\overline{\mathsf{RealSig}}(j) := \{\mathsf{RealSig}(j)_k\}_{k \in \mathbb{N}}$ and $\overline{\mathsf{IdealSig}}(j) := \{\mathsf{IdealSig}(j)_k\}_{k \in \mathbb{N}}$. Note that the hiding operation prevents the environment from learning the signing key.

Theorem 6.2. *Let $j \in SID$ be given. Suppose that $\langle \mathsf{KeyGen}_j, \mathsf{Sign}_j, \mathsf{Verify}_j \rangle$ is a complete and EUF-CMA secure signature scheme. Then* $\overline{\mathsf{RealSig}}(j) \leq_{\mathsf{neg,pt}}^{\{\mathsf{fail}_j\}} \overline{\mathsf{IdealSig}}(j)$.

To prove Theorem 6.2, we show that, for every time point t, the environment cannot distinguish $\mathsf{RealSig}(j)_k$ from $\mathsf{IdealSig}(j)_k$ with high probability between time t and $t + q(k)$, where q is a polynomial. This holds even when the task $\{fail_j\}$ is not scheduled in the interval $[t, t + q]$. The interesting case is when j is awakened *after* time t. That implies the *failed* flag is never set and $\mathsf{SigFunc}(k, j)$ uses $history$ to reject forgeries.

We use the the EUF-CMA assumption to obtain a bound on the distinguishing probability of any environment. Essentially, we build a forger that emulates the execution of our various task-PIOAs under some valid schedule. When the environment interacts with the Signer and Verifier automata, this forger uses the signature oracle and verification algorithm in the EUF-CMA game. Moreover, the success probability of this forger is maximized over all environments satisfying a particular polynomial bound. (Note that, given polynomial p and security parameter k, there are only a finite number of $p(k)$-bounded environments.) Applying the definition of EUF-CMA security, we obtain the desired negligible bound on distinguishing probability.

Proof of Theorem 6.2. Unwinding the definition of $\leq_{\mathsf{neg,pt}}^{\{\mathsf{fail}_j\}}$, we need to show the following: for every polynomials p and q, there is a negligible function ϵ such that, for every $k \in \mathbb{N}$, $t \in \mathbb{R}_{\geq 0}$, $p(k)$-bounded environment Env for $\mathsf{RealSig}(j)_k$, and valid schedule τ_1 for $\mathsf{RealSig}(j)_k \| \mathsf{Env}$ for the interval $[0, t + q(k)]$, there is a valid schedule τ_2 for $\mathsf{IdealSig}(j)_k, \| \mathsf{Env}$ such that

(i) $\mathbf{P}_{\mathsf{acc}}(\mathsf{RealSig}(j)_k \| \mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_1))$ is the same as $\mathbf{P}_{\mathsf{acc}}(\mathsf{IdealSig}(j)_k \| \mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_2))$;

(ii) $\mathsf{lstate}_{\mathsf{Env}}(\mathsf{RealSig}(j)_k \| \mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_1))$ is the same as $\mathsf{lstate}_{\mathsf{Env}}(\mathsf{IdealSig}(j)_k \| \mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_2))$;

(iii) $\mathsf{proj}_{\mathsf{Env}}(\tau_1) = \mathsf{proj}_{\mathsf{Env}}(\tau_2)$;

(iv) τ_2 does not contain any pairs of the form $\langle \mathsf{fail}_j, t_i \rangle$ where $t_i \geq t$;

(v) $\mathbf{P}_{\mathsf{acc}}(\mathsf{RealSig}(j)_k \| \mathsf{Env}, \tau_1)$ is at most $\epsilon(k)$ away from $\mathbf{P}_{\mathsf{acc}}(\mathsf{IdealSig}(j)_k \| \mathsf{Env}, \tau_2)$.

Let polynomial p and q be given. We need to obtain a negligible ϵ bound that makes all the conditions above satisfied for every k, t, $p(k)$-bounded Env, valid τ_1, and some corresponding τ_2.

Fix t_l and t_r to be time points such that $[t_l, t_r] = \{t \in \mathbb{T} | j \in \mathsf{alive}(t)\}$. So, we know that both $\mathsf{RealSig}(j)_k$ and $\mathsf{IdealSig}(j)_k$ are dormant outside the interval $[t_l, t_r]$.

First consider the cases in which $t_l < t$. We obtain τ_2 by inserting $\langle \{\mathsf{fail}_j\}, t_l \rangle$ immediately after $\langle \mathsf{tick}, t_l \rangle$. This sets the *failed* flag in $\mathsf{SigFunc}(k, j)$ to true immediately after *awake* becomes true. Notice that, if *failed* $=$ true, the verify transition bypasses the check $m \in \mathit{history}$ (Figure 5). In other words, $\mathsf{SigFunc}(k, j)$ answers verify requests in exactly the same way as $\mathsf{Verifier}(k, j)$, using the Verify algorithm only. Furthermore, it is easy to check that *failed* remains true as long as $\mathsf{SigFunc}(k, j)$ is alive. Therefore, $\mathsf{IdealSig}(j)_k$ has exactly the same visible behavior as $\mathsf{RealSig}(j)_k$ and Conditions (i) through (v) above are satisfied if we choose $\epsilon(k) = 0$, for every k, $p(k)$-bounded Env and valid τ_1.

Now, consider the cases in which $t \leq t_l$. Set $\tau_2 := \tau_1$. Since both $\mathsf{RealSig}(j)_k$ and $\mathsf{IdealSig}(j)_k$ are dormant during $[0, t]$, Conditions (i) and (ii) must hold. Condition (iii) is immediate and Condition (iv) holds because fail_j is not a task of $\mathsf{RealSig}(j)_k$. It remains to argue that there exists a negligible function ϵ such that Condition (v) is satisfied.

To this purpose, we rely on the EUF-CMA security of Sig. We however do not need to bound the success probability of one specific forger, as in the EUF-CMA definition, but the success probability of all forgers that satisfy fixed polynomial p and q bounds, for every time t and schedule τ_1.

For every $k \in \mathbb{N}$, we define a time $(t_{max})_k \leq t_l$, a $p(k)$-bounded environment $(\mathsf{Env}_{max})_k$ for $\mathsf{RealSig}_k$, and a valid schedule $(\tau_{1max})_k$ for $\mathsf{RealSig}_k \| (\mathsf{Env}_{max})_k$ for the time interval $[0, (t_{max})_k + q(k)]$, with the following property: for every time $t \leq t_l$, every $p(k)$-bounded environment Env for $\mathsf{RealSig}_k$, and every valid schedule τ_1 for $\mathsf{RealSig}_k \| \mathsf{Env}$ for the interval $[0, t + q(k)]$, we have:

$$\begin{aligned} &| \mathbf{P}_{\mathsf{acc}}(\mathsf{RealSig}(j)_k \| \mathsf{Env}, \tau_1) - \mathbf{P}_{\mathsf{acc}}(\mathsf{IdealSig}(j)_k \| \mathsf{Env}, \tau_1)| \\ &\leq | \mathbf{P}_{\mathsf{acc}}(\mathsf{RealSig}(j)_k \| (\mathsf{Env}_{max})_k, (\tau_{1max})_k) - \mathbf{P}_{\mathsf{acc}}(\mathsf{IdealSig}(j)_k \| (\mathsf{Env}_{max})_k, (\tau_{1max})_k)|. \end{aligned}$$

To see that such a $(t_{\max})_k$, $(\mathsf{Env}_{max})_k$ and $(\tau_{1max})_k$ exist, it is enough to observe that there are only a finite number of times, environments and schedules respecting the t_l, $p(k)$ and $q(k)$ bounds (up to isomorphism).

This means that is enough to show the existence of a negligible function ϵ such that, for every $k \in \mathbb{N}$, we have:

$$| \mathbf{P}_{\mathsf{acc}}(\mathsf{RealSig}(j)_k \| (\mathsf{Env}_{max})_k, (\tau_{1max})_k) - \mathbf{P}_{\mathsf{acc}}(\mathsf{IdealSig}(j)_k \| (\mathsf{Env}_{max})_k, (\tau_{1max})_k)| \leq \epsilon(k).$$

Since Sig is complete, we observe that, for every value of k, the difference of acceptance probabilities of the two automata compared in Condition (v) can only be non-zero if $(\mathsf{Env}_{max})_k$ succeeds in producing a forged signature (that is, a valid signature for a message that was not signed by the Sign or SigFunc automata before) and in having this signature rejected when the verify and respVer actions of SigFunc execute.

We now use each $(\mathsf{Env}_{max})_k$ and $(\tau_{1max})_k$ to define a probabilistic polynomial-time (non-uniform) forger $G = \{G_k\}_{k \in \mathbb{N}}$ for Sig, in such a way that G_k essentially emulates an execution of the automaton $\mathsf{IdealSig}(j)_k \| (\mathsf{Env}_{max})_k$ with schedule $(\tau_{1max})_k$.

More precisely, G_k successively reads all the tasks in the schedule $(\tau_{1max})_k$, and uses them to internally emulate an execution of $\mathsf{IdealSig}(j)_k \| (\mathsf{Env}_{max})_k$, up to the following exceptions:

1. when the $\{\mathsf{verKey}(*)\}$ task has to be emulated, G_k replaces the verification algorithm obtained when emulating the $\{\mathsf{chooseKeys}\}$ task with the one provided by Sig in the EUF-CMA game, and
2. when the $\{\mathsf{sign}(*,*)\}$ task has to be emulated, G_k obtains signatures by using the signing oracle available in the EUF-CMA game.

Furthermore, G_k stores a list of all messages that the emulated $(\mathsf{Env}_{max})_k$ asked to sign, and checks whether $(\mathsf{Env}_{max})_k$ ever asks for the verification of a message with a valid signature that is not in the list. If such a signature is produced, G_k outputs it as a forgery.

We observe that this emulation process is polynomial time-bounded because all transitions of the emulated systems are polynomial time-bounded, the total running time of the system is bounded by $t_l + q(k)$, and Condition (iii) on the overall bound of automata guarantees that no more than a polynomial number of transitions are performed per time unit.

We also observe that the two proposed exceptions in the emulation of the execution of $\mathsf{IdealSig}(j)_k \| (\mathsf{Env}_{max})_k$ do not change the distribution of the messages that $(\mathsf{Env}_{max})_k$ sees, since the verification algorithm used by G_k is generated in the same way as KeyGen generates it, and since the message signatures are also produced in a valid way. Therefore, it is with the same probability that the environment distinguishes the two systems it is interacting with (that is, by producing a forgery early enough) in a real execution of the different automata and in the version emulated by G.

Now, the assumption that Sig is EUF-CMA secure guarantees that there exists a negligible function ϵ bounding the success probability of G. Selecting this function ϵ completes our proof. □

7 Composition Theorems

In practice, cryptographic services are seldom used in isolation. Most likely, different types of services operate in conjunction, interacting with each other and with multiple protocol participants. For example, a participant may submit a document to an encryption service to obtain a ciphertext, which is later submitted to a timestamping service. In such situations, it is important that the services are provably secure even in the context of composition.

In this section, we consider two types of composition. The first, *parallel composition*, is a combination of services that are active at the same time and may interact with each other. Given a polynomially bounded collection of real services such that each real service implement some ideal service, the parallel composition of the real services is guaranteed to implement that of the ideal services.

The second type, *sequential composition*, is a combination of services that are active in succession. The interaction between two distinct services is much more limited in this setting, because the earlier one must have finished execution before the later one comes online. An example of such a collection is the signature services in the timestamping protocol of [5, 4], where each service is replaced by the next at regular intervals.

As in the parallel case, we prove that the sequential composition of real services implements the sequential composition of ideal services. We are able to relax the restriction on the number of components from polynomial to exponential.[2] This highlights a unique aspect of our implementation relation: essentially, we walk down the real time line and, at every point t, we focus on a polynomial length interval starting from t.

Parallel Composition Using a standard hybrid argument, we show that the relation $\leq^{F}_{p,q,\epsilon}$ (cf. Definition 5.1) is preserved under polynomial parallel composition, with some appropriate adjustment to the environment complexity bound and to the error in acceptance probability.

Theorem 7.1. *Let $b \in \mathbb{N}$ be given and, for each $1 \leq i \leq b$, let $\mathcal{A}_i^1$ and $\mathcal{A}_i^2$ be comparable task-PIOAs and let F_i be a set of tasks of $\mathcal{A}_i^2$. Let $\hat{F}$ denote $\bigcup_{i=1}^{b} F_i$. Suppose there exists a non-decreasing function $r : \mathbb{N} \to \mathbb{N}$ such that, for all i, both $\mathcal{A}_i^1$ and $\mathcal{A}_i^2$ are $r(i)$-bounded. Suppose further that $\mathcal{A}_1^{\alpha_1}, \ldots, \mathcal{A}_b^{\alpha_b}$ are pairwise compatible for any combination of $\alpha_i \in \{1, 2\}$.*

Let $p, p', q \in \mathbb{N}$ and $\epsilon, \epsilon' \in \mathbb{R}_{\geq 0}$ be given, and assume the following.
(1) $p = c_{\mathsf{comp}} \cdot (b \cdot r(b) + p')$, where c_{comp} is the constant factor for composing task-PIOAs in parallel.
(2) $\epsilon' = b \cdot \epsilon$.
(3) For all i, $\mathcal{A}_i^1 \leq^{F_i}_{p,q,\epsilon} \mathcal{A}_i^2$.

[2]In our model, it is not meaningful to exceed an exponential number of components, because the length of the description of each component is polynomially bounded.

Then we have $\|_{i=1}^{b}\mathcal{A}_i^1 \leq_{p',q,\epsilon'}^{\hat{F}} \|_{i=1}^{b}\mathcal{A}_i^2$.

Proof. Let $t \in \mathbb{R}_{\geq 0}$ be given. Let $\mathsf{Env} = \mathsf{Env}'\|\mathsf{Clock}$ be a p'-bounded environment and let τ_0 be a valid timed task schedule for $\|_{i=1}^{b}\mathcal{A}_i^1\|\mathsf{Env}$ for the interval $[0, t+q]$.

For each $0 \leq i \leq b$, let H_i denote $\mathcal{A}_1^2\|\ldots\|\mathcal{A}_i^2\|\mathcal{A}_{i+1}^1\|\ldots\|\mathcal{A}_b^1$. In particular, $H_0 = \|_{i=1}^{b}\mathcal{A}_i^1$ and $H_b = \|_{i=1}^{b}\mathcal{A}_i^2$. Similarly, let

$$\mathsf{Env}_i := \mathcal{A}_1^2\|\ldots\|\mathcal{A}_{i-1}^2\|\mathcal{A}_{i+1}^1\|\ldots\|\mathcal{A}_b^1\|\mathsf{Env}$$

for each $1 \leq i \leq b$. Note that every Env_i is p-bounded and is an environment for $\mathcal{A}_i^1$ and $\mathcal{A}_i^2$. In fact, we have $H_{i-1}\|\mathsf{Env} = \mathcal{A}_i^1\|\mathsf{Env}_i$ and $H_i\|\mathsf{Env} = \mathcal{A}_i^2\|\mathsf{Env}_i$.

Since $\mathcal{A}_1^1 \leq_{p,q,\epsilon}^{F_1} \mathcal{A}_1^2$ and τ_0 is a valid schedule for $\mathcal{A}_1^1\|\mathsf{Env}_1$, we may choose a valid schedule τ_1 for $\mathcal{A}_1^2\|\mathsf{Env}_1$ for the interval $[0, t+q]$ such that

(i) $\mathbf{P}_{\mathsf{acc}}(\mathcal{A}_1^1\|\mathsf{Env}_1, \mathsf{trunc}_{\geq t}(\tau_0)) = \mathbf{P}_{\mathsf{acc}}(\mathcal{A}_1^2\|\mathsf{Env}_1, \mathsf{trunc}_{\geq t}(\tau_1))$;

(ii) $\mathsf{lstate}_{\mathsf{Env}_1}(\mathcal{A}_1^1\|\mathsf{Env}_1, \mathsf{trunc}_{\geq t}(\tau_0)) = \mathsf{lstate}_{\mathsf{Env}_1}(\mathcal{A}_1^2\|\mathsf{Env}_1, \mathsf{trunc}_{\geq t}(\tau_1))$;

(iii) $\mathsf{proj}_{\mathsf{Env}_1}(\tau_0) = \mathsf{proj}_{\mathsf{Env}_1}(\tau_1)$;

(iv) τ_1 does not contain any pairs of the form $\langle T_i, t_i\rangle$ where $T_i \in F_1$ and $t_i \geq t$;

(v) $|\,\mathbf{P}_{\mathsf{acc}}(\mathcal{A}_1^1\|\mathsf{Env}_1, \tau_0) - \mathbf{P}_{\mathsf{acc}}(\mathcal{A}_1^2\|\mathsf{Env}_1, \tau_1)| \leq \epsilon$.

Repeating this argument, we choose valid schedules $\tau_2, \ldots, \tau_b$ for $H_2\|\mathsf{Env}, \ldots, H_b\|\mathsf{Env}$, respectively, all satisfying the appropriate five conditions. By Condition (i), we have

$$\mathbf{P}_{\mathsf{acc}}(H_0\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_0)) = \mathbf{P}_{\mathsf{acc}}(H_1\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_1)) = \ldots = \mathbf{P}_{\mathsf{acc}}(H_b\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_b)).$$

Also, since Env is part of every Env_i, Condition (ii) guarantees that

$$\mathsf{lstate}_{\mathsf{Env}}(H_0\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_0)) = \mathsf{lstate}_{\mathsf{Env}}(H_b\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_b)).$$

Similarly, Condition (iii) guarantees that $\mathsf{proj}_{\mathsf{Env}}(\tau_0) = \mathsf{proj}_{\mathsf{Env}}(\tau_b)$.

Using both Conditions (iii) and (iv), we can infer that τ_b does not contain any pairs of the form $\langle T_i, t_i\rangle$ where $T_i \in \hat{F} = \bigcup_{i=1}^{b} F_i$ and $t_i \geq t$. Finally,

$$\begin{aligned}
&|\,\mathbf{P}_{\mathsf{acc}}(\|_{i=1}^{b}\mathcal{A}_i^1\|\mathsf{Env}, \tau_0) - \mathbf{P}_{\mathsf{acc}}(\|_{i=1}^{b}\mathcal{A}_i^2\|\mathsf{Env}, \tau_b)| \\
&\leq |\,\mathbf{P}_{\mathsf{acc}}(H_0\|\mathsf{Env}, \tau_0) - \mathbf{P}_{\mathsf{acc}}(H_1\|\mathsf{Env}, \tau_1)| + \ldots \\
&\quad + |\,\mathbf{P}_{\mathsf{acc}}(H_i\|\mathsf{Env}, \tau_i) - \mathbf{P}_{\mathsf{acc}}(H_{i+1}\|\mathsf{Env}, \tau_{i+1})| + \ldots \\
&\quad + |\,\mathbf{P}_{\mathsf{acc}}(H_{b-1}\|\mathsf{Env}, \tau_{b-1}) - \mathbf{P}_{\mathsf{acc}}(H_b\|\mathsf{Env}, \tau_b)| \\
&\leq b \cdot \epsilon = \epsilon'.
\end{aligned}$$

□

Using Theorem 7.1, it is not hard to prove that $\leq_{\mathsf{neg,pt}}^{\bar{F}}$ is preserved under polynomial composition.

Theorem 7.2 (Parallel Composition Theorem for $\leq_{\mathsf{neg,pt}}^{\bar{F}}$). *Let two sequences of task-PIOA families $\bar{\mathcal{A}}_1^1, \bar{\mathcal{A}}_2^1, \ldots$ and $\bar{\mathcal{A}}_1^2, \bar{\mathcal{A}}_2^2, \ldots$ be given, with $\bar{\mathcal{A}}_i^1$ comparable to $\bar{\mathcal{A}}_i^2$ for all i. Assume that $\bar{\mathcal{A}}_1^{\alpha_1}, \bar{\mathcal{A}}_2^{\alpha_2}, \ldots$ are pairwise compatible for any combination of $\alpha_i \in \{1, 2\}$. For each i, let $\bar{F}_i$ be a family of sets such that $(\bar{F}_i)_k$ is a set of tasks of $(\bar{\mathcal{A}}_i^2)_k$ for every k.*

Suppose there exist polynomials $r, s : \mathbb{N} \to \mathbb{N}$ such that, for all i, k, both $(\bar{\mathcal{A}}_i^1)_k$ and $(\bar{\mathcal{A}}_i^2)_k$ are bounded by $s(k) \cdot r(i)$. Assume that r is non-decreasing and

$$\forall p, q\; \exists \epsilon\; \forall i\; \bar{\mathcal{A}}_i^1 \leq_{p,q,\epsilon}^{\bar{F}_i} \bar{\mathcal{A}}_i^2, \tag{1}$$

where p, q are polynomials and ϵ is a negligible function. (This is a strengthening of the statement $\forall i\; \bar{\mathcal{A}}_i^1 \leq_{\mathsf{neg,pt}}^{\bar{F}_i} \bar{\mathcal{A}}_i^2$.) Let b be any polynomial. For each k, let $(\widehat{\mathcal{A}}^1)_k$ denote $(\bar{\mathcal{A}}_1^1)_k\|\ldots\|(\bar{\mathcal{A}}_{b(k)}^1)_k$. Similarly for $(\widehat{\mathcal{A}}^2)_k$. Also, let $(\hat{F})_k$ denote $\bigcup_{i=1}^{b(k)}(\bar{F}_i)_k$. Then we have $\widehat{\mathcal{A}}^1 \leq_{\mathsf{neg,pt}}^{\hat{F}} \widehat{\mathcal{A}}^2$.

Proof. By the definition of $\leq^{\hat{F}}_{\mathsf{neg,pt}}$, we need to prove the following: $\forall p', q \; \exists \epsilon' \; \widehat{\mathcal{A}}^1 \leq^{\hat{F}}_{p',q,\epsilon'} \widehat{\mathcal{A}}^2$, where p', q are polynomials and ϵ' is a negligible function. Let polynomials p' and q be given and define $p := c_{\mathsf{comp}} \cdot (b \cdot (r \circ b) + p')$, where c_{comp} is the constant factor for composing task-PIOAs in parallel. Now choose ϵ using p, q, and Assumption (1). Define $\epsilon' := b \cdot \epsilon$.

Let $k \in \mathbb{N}$ be given. We need to prove $(\widehat{\mathcal{A}}^1)_k \leq^{(\hat{F})_k}_{p'(k),q(k),\epsilon'(k)} (\widehat{\mathcal{A}}^2)_k$. That is,

$$(\bar{\mathcal{A}}^1_1)_k \| \ldots \| (\bar{\mathcal{A}}^1_{b(k)})_k \leq^{\bigcup_{i=1}^{b(k)} (\bar{F}_i)_k}_{p'(k),q(k),\epsilon'(k)} (\bar{\mathcal{A}}^2_1)_k \| \ldots \| (\bar{\mathcal{A}}^2_{b(k)})_k.$$

For every i, we know that $(\bar{\mathcal{A}}^1_i)_k$ and $(\bar{\mathcal{A}}^2_i)_k$ are bounded by $(s(k) \cdot r)(i)$. Also, by the choice of ϵ, we have $(\bar{\mathcal{A}}^1_i)_k \leq^{(\bar{F}_i)_k}_{p(k),q(k),\epsilon(k)} (\bar{\mathcal{A}}^2_i)_k$ for all i. Therefore, we may apply Theorem 7.1 to conclude that $(\widehat{\mathcal{A}}^1)_k \leq^{\bigcup_{i=1}^{b(k)} (\bar{F}_i)_k}_{p'(k),q(k),\epsilon'(k)} (\widehat{\mathcal{A}}^2)_k$. This completes the proof. □

Sequential Composition We now treat the more interesting case, namely, exponential sequential composition. The first challenge is to formalize the notion of sequentiality. On a syntactic level, all components in the collection are combined using the parallel composition operator. To capture the idea of successive invocation, we introduce some auxiliary notions. Intuitively, we distinguish between *active* and *dormant* entities. Active entities may perform actions and store information in memory. Dormant entities have no available memory and do not enable locally controlled actions.[3] In Definition 7.3, we formalize the idea that an entity $\mathcal{A}$ may be invoked and terminated by some other entity $\mathcal{B}$. Then we introduce sequentiality in Definition 7.5.

Definition 7.3. *Let $\mathcal{A}$ and $\mathcal{B}$ be pairwise compatible task-PIOAs and let reals $t_1 \leq t_2$ be given. We say that $\mathcal{A}$ is* restricted to the interval $[t_1, t_2]$ by $\mathcal{B}$ *if:*

- *for any $t < t_1$, environment Env for $\mathcal{A} \| \mathcal{B}$ of the form $\mathsf{Env}' \| \mathsf{Clock}$, valid schedule τ for $\mathcal{A} \| \mathcal{B} \| \mathsf{Env}$ for $[0, t]$, and state s reachable under τ, no locally controlled actions of $\mathcal{A}$ are enabled in s, and $s.v = \bot$ for every variable v of $\mathcal{A}$.*
- *the same for all $t > t_2$.*

Lemma 7.4 below states the intuitive fact that no environment can distinguish two entities during an interval in which both entities are dormant.

Lemma 7.4. *Suppose $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable task-PIOAs that are both restricted to the interval $[t_1, t_2]$ by $\mathcal{B}$. Let Env be an environment for both $\mathcal{A}_1 \| \mathcal{B}$ and $\mathcal{A}_2 \| \mathcal{B}$ and of the form $\mathsf{Env}' \| \mathsf{Clock}$. Let $t \in \mathbb{R}_{\geq 0}$ and $q \in \mathbb{N}$ be given. Suppose we have valid schedule τ_1 for $\mathcal{A}_1 \| \mathcal{B} \| \mathsf{Env}$ for the interval $[0, t+q]$ and valid schedule τ_2 for $\mathcal{A}_2 \| \mathcal{B} \| \mathsf{Env}$ for the interval $[0, t+q]$, satisfying:*

- $\mathbf{P}_{\mathsf{acc}}(\mathcal{A}_1 \| \mathcal{B} \| \mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_1)) = \mathbf{P}_{\mathsf{acc}}(\mathcal{A}_2 \| \mathcal{B} \| \mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_2))$*;*
- $\mathsf{lstate}_{\mathcal{B} \| \mathsf{Env}}(\mathcal{A}_1 \| \mathcal{B} \| \mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_1)) = \mathsf{lstate}_{\mathcal{B} \| \mathsf{Env}}(\mathcal{A}_2 \| \mathcal{B} \| \mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_2))$*;*
- $\mathsf{proj}_{\mathcal{B} \| \mathsf{Env}}(\tau_1) = \mathsf{proj}_{\mathcal{B} \| \mathsf{Env}}(\tau_2)$.

Assume further that either $t_2 < t$ or $t_1 > t + q$. Then $\mathbf{P}_{\mathsf{acc}}(\mathcal{A}_1 \| \mathcal{B} \| \mathsf{Env}, \tau_1) = \mathbf{P}_{\mathsf{acc}}(\mathcal{A}_2 \| \mathcal{B} \| \mathsf{Env}, \tau_2)$).

Proof. First we consider the case $t_2 < t$. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are restricted by $\mathcal{B}$ to the interval $[t_1, t_2]$, neither of them enables any output actions during the interval $[t, t+q]$. By assumption, τ_1 and τ_2 agree on the tasks of $\mathcal{B} \| \mathsf{Env}$ and the state distributions of $\mathcal{B} \| \mathsf{Env}$ just before time t are identical in the two experiments. Therefore, the probability that Env outputs acc during $[t, t+q]$ must be identical in the two experiments. We also have the assumption that Env outputs acc with the same probability during $[0, t)$, therefore the acceptance probabilities are the same for the entire interval $[0, t]$.

Similarly, if $t_1 > t + q$, then neither $\mathcal{A}_1$ nor $\mathcal{A}_2$ enables any output actions during the interval $[t, t+q]$. Then we follow the same argument as above. □

Definition 7.5 (Sequentiality). *Let $\mathcal{B}, \mathcal{A}_1, \mathcal{A}_2, \ldots$ be pairwise compatible task-PIOAs. We say that $\mathcal{A}_1, \mathcal{A}_2, \ldots$ are* sequential under $\mathcal{B}$ *if there exist reals $0 \leq t_1 < t_2 < \ldots$ such that: for all i, $\mathcal{A}_i$ is restricted to $[t_i, t_{i+1}]$ by $\mathcal{B}$.*

[3]For technical reasons, dormant entities must synchronize on input actions. Some inputs cause dormant entities to become active, while all others are trivial loops on the null state.

Note that each $\mathcal{A}_i$ may overlap with $\mathcal{A}_{i+1}$ at the boundary time t_{i+1}. Now we are ready to state the sequential composition theorems.

Theorem 7.6. *Let $\mathcal{A}_1^1, \mathcal{A}_2^1, \ldots$ and $\mathcal{A}_1^2, \mathcal{A}_2^2, \ldots$ be two sequences of task-PIOAs such that $\mathcal{A}_i^1$ and $\mathcal{A}_i^2$ are comparable for every i. Assume that $\mathcal{A}_1^{\alpha_1}, \mathcal{A}_2^{\alpha_2}, \ldots$ are pairwise compatible for any combination of $\alpha_i \in \{1,2\}$. Also, let $L, \hat{p} \in \mathbb{N}$ be given and let $\mathcal{B}$ be a task-PIOA such that both $\mathcal{B}\|(\|_{i=1}^L \mathcal{A}_i^1)$ and $\mathcal{B}\|(\|_{i=1}^L \mathcal{A}_i^2)$ are $\hat{p}$-bounded. Assume that both $\mathcal{A}_1^1, \ldots, \mathcal{A}_L^1$ and $\mathcal{A}_1^2, \ldots, \mathcal{A}_L^2$ are sequential under $\mathcal{B}$ for the same sequence of reals $t_1 < \ldots < t_{L+1}$.*

Let $p, q \in \mathbb{N}$ and $\epsilon \in \mathbb{R}_{\geq 0}$ be given. Suppose there are sets of tasks F_i, $1 \leq i \leq L$, such that $\mathcal{A}_i^1 \leq_{p,q,\epsilon}^{F_i} \mathcal{A}_i^2$ for all i. Let $\hat{F}$ denote $\bigcup_{i=1}^L F_i$. Let b denote the largest number such that b consecutive t_i's fall into a single closed interval of length q. (Such b must exist and is between 1 and L). Let $p' \in \mathbb{N}$ and $\epsilon' \in \mathbb{R}_{\geq 0}$ be given, with $\epsilon' \geq (b+2) \cdot \epsilon$ and $p \geq c_{\mathsf{comp}} \cdot (\hat{p} + p')$ (where c_{comp} is the constant factor for parallel composition). Then we have $\mathcal{B}\|(\|_{i=1}^L \mathcal{A}_i^1) \leq_{p',q,\epsilon'}^{\hat{F}} \mathcal{B}\|(\|_{i=1}^L \mathcal{A}_i^2)$.

In the statement of Theorem 7.6, the error in acceptance probability increases by a factor of $b+2$, where b is the largest number of components that may be active in a closed time interval of length q. For example, if the life time of each component is $\frac{q}{3}$, then b is 5.[4] This is the key difference between parallel composition and sequential composition: for the former, error increases with the total number of components (namely, L), and hence no more than a polynomial number of components can be tolerated. In the sequential case, L may be exponential, as long as b remains small. The proof of Theorem 7.6 involves a standard hybrid argument for active components, while dormant components are replaced without affecting the difference in acceptance probabilities.

Proof of Theorem 7.6. Let $t \in \mathbb{R}_{\geq 0}$ be given. Let $\mathsf{Env} = \mathsf{Env}'\|\mathsf{Clock}$ be a p'-bounded environment and let τ_0 be a valid timed task schedule for $\mathcal{B}\|(\|_{i=1}^L \mathcal{A}_i^1)\|\mathsf{Env}$ for the interval $[0, t+q]$. We need to find τ_L for $\mathcal{B}\|(\|_{i=1}^L \mathcal{A}_i^2)\|\mathsf{Env}$ such that

(i) $\mathbf{P}_{\mathsf{acc}}(\|_{i=1}^L \mathcal{A}_i^1\|B\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_0)) = \mathbf{P}_{\mathsf{acc}}(\|_{i=1}^L \mathcal{A}_i^2\|B\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_L))$;

(ii) $\mathsf{lstate}_{\mathsf{Env}}(\|_{i=1}^L \mathcal{A}_i^1\|B\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_0)) = \mathsf{lstate}_{\mathsf{Env}}(\|_{i=1}^L \mathcal{A}_i^2\|B\|\mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_L))$;

(iii) $\mathsf{proj}_{\mathsf{Env}}(\tau_0) = \mathsf{proj}_{\mathsf{Env}}(\tau_L)$;

(iv) τ_L does not contain any pairs of the form $\langle T_i, t_i \rangle$ where $T_i \in \hat{F}$ and $t_i \geq t$;

(v) $|\,\mathbf{P}_{\mathsf{acc}}(\|_{i=1}^L \mathcal{A}_i^1\|B\|\mathsf{Env}, \tau_0) - \mathbf{P}_{\mathsf{acc}}(\|_{i=1}^L \mathcal{A}_i^2\|B\|\mathsf{Env}, \tau_L)| \leq \epsilon'$.

Without loss of generality, assume there is an index i such that $[t_i, t_{i+1}]$ intersects with $[t, t+q]$. Let l be the smallest such index. Recall from the assumptions that at most b consecutive t_i's fall into a closed interval of length q. Therefore, we know that $t_{l-1} < t$ and $t_{l+b} > t+q$.

The rest of the proof proceeds as in the proof of Theorem 7.1. Namely, we define

$$\mathsf{Env}_i := \mathcal{A}_1^2\|\ldots\|\mathcal{A}_{i-1}^2\|\mathcal{A}_{i+1}^1\|\ldots\|\mathcal{A}_b^1\|\mathcal{B}\|\mathsf{Env}$$

for each $1 \leq i \leq L$. Note that Env_i is p-bounded, therefore we may choose τ_{i+1} using τ_i and the assumption that $\mathcal{A}_i^1 \leq_{p,q,\epsilon}^{F_i} \mathcal{A}_i^2$. Since Env is part of Env_i for every i, Conditions (i) through (iii) are clearly satisfied at every replacement step. Condition (iv) is satisfied because the following hold at every step i.

- The new task schedule τ_{i+1} does not contain tasks from F_{i+1}.
- Condition (iii) guarantees that τ_{i+1} does not contain tasks from $\bigcup_{j=1}^i F_j$.

Finally, we consider Condition (v). There are two cases. If $i < l-1$ or $i \geq l+b$, then we can apply Lemma 7.4 to conclude that $\mathbf{P}_{\mathsf{acc}}(\mathcal{A}_i^1\|\mathsf{Env}_i, \tau_i)$ in fact equals $\mathbf{P}_{\mathsf{acc}}(\mathcal{A}_i^2\|\mathsf{Env}_i, \tau_{i+1})$. Otherwise, $\mathbf{P}_{\mathsf{acc}}(\mathcal{A}_i^1\|\mathsf{Env}_i, \tau_i)$ and $\mathbf{P}_{\mathsf{acc}}(\mathcal{A}_i^2\|\mathsf{Env}_i, \tau_{i+1})$ differ by at most ϵ. Summing over all indices i, we have $|\,\mathbf{P}_{\mathsf{acc}}(\|_{i=1}^L \mathcal{A}_i^1\|B\|\mathsf{Env}, \tau_0) - \mathbf{P}_{\mathsf{acc}}(\|_{i=1}^L \mathcal{A}_i^2\|B\|\mathsf{Env}, \tau_L)|$
$\leq (b+2) \cdot \epsilon = \epsilon'$. □

Using Theorem 7.6, it is straightforward to prove the sequential composition theorem for $\leq_{\mathsf{neg,pt}}^{\bar{F}}$.

[4]Recall that two components may be active simultaneously at the boundary time.

Theorem 7.7 (Sequential Composition Theorem for $\leq^{\bar{F}}_{\mathsf{neg,pt}}$). *Let two sequences of task-PIOA families $\bar{\mathcal{A}}^1_1, \bar{\mathcal{A}}^1_2, \ldots$ and $\bar{\mathcal{A}}^2_1, \bar{\mathcal{A}}^2_2, \ldots$ be given, with $\bar{\mathcal{A}}^1_i$ comparable to $\bar{\mathcal{A}}^2_i$ for all i. Assume that $\bar{\mathcal{A}}^{\alpha_1}_1, \bar{\mathcal{A}}^{\alpha_2}_2, \ldots$ are pairwise compatible for any combination of $\alpha_i \in \{1,2\}$. For each i, let $\bar{F}_i$ be a family of sets such that $(\bar{F}_i)_k$ is a set of tasks of $(\bar{\mathcal{A}}^2_i)_k$ for every k. Let $L : \mathbb{N} \to \mathbb{N}$ be an exponential function and, for each k, let $(\widehat{\mathcal{A}}^1)_k$ denote $(\bar{\mathcal{A}}^1_1)_k \| \ldots \| (\bar{\mathcal{A}}^1_{L(k)})_k$. Similarly for $(\widehat{\mathcal{A}}^2)_k$. Also, let $(\hat{F})_k$ denote $\bigcup_{i=1}^{L(k)} (\bar{F}_i)_k$.*

Let $\hat{p}$ be a polynomial and let $\bar{\mathcal{B}}$ be a task-PIOA family such that both $\bar{\mathcal{B}} \| \widehat{\mathcal{A}}^1$ and $\bar{\mathcal{B}} \| \widehat{\mathcal{A}}^2$ are $\hat{p}$-bounded. Suppose there exist a sequence of positive reals $t_1 < t_2 < \ldots$ such that, for each k, both $(\bar{\mathcal{A}}^1_1)_k, \ldots, (\bar{\mathcal{A}}^1_{L(k)})_k$ and $(\bar{\mathcal{A}}^2_1)_k, \ldots, (\bar{\mathcal{A}}^2_{L(k)})_k$ are sequential under $\mathcal{B}_k$ for the sequence $t_1 < \ldots < t_{L(k)+1}$. Assume there is a constant real number c such that consecutive t_i's are at least c apart.

Suppose that, for every pair of polynomials $\langle p, q \rangle$, there exists negligible function ϵ such that $\bar{\mathcal{A}}^1_i \leq^{\bar{F}_i}_{p,q,\epsilon} \bar{\mathcal{A}}^2_i$ for all i. Then we have $\bar{\mathcal{B}} \| \widehat{\mathcal{A}}^1 \leq^{\hat{F}}_{\mathsf{neg,pt}} \bar{\mathcal{B}} \| \widehat{\mathcal{A}}^2$.

Proof. Let polynomials p', q be given and define $p := c_{\mathsf{comp}} \cdot (\hat{p} + p')$, where c_{comp} is the constant factor for composing task-PIOAs in parallel. Choose ϵ from p, q according to the assumption of the theorem. For each k, let $b(k)$ be the ceiling of $\frac{q(k)}{c} + 1$. (The choice of $b(k)$ ensures that at most $b(k)$ consecutive t_i's fall within any interval of length at most $q(k)$. This is necessary in order to apply Theorem 7.6.) Since c is constant, b is a polynomial. Define $\epsilon' := b \cdot \epsilon$.

For every $k \in \mathbb{N}$, we apply Theorem 7.6 to conclude that

$$\bar{\mathcal{B}}_k \| (\bar{\mathcal{A}}^1_1)_k \| \ldots \| (\bar{\mathcal{A}}^1_{L(k)})_k \leq^{(\hat{F})_k}_{p'(k),q(k),\epsilon'(k)} \bar{\mathcal{B}}_k \| (\bar{\mathcal{A}}^2_1)_k \| \ldots \| (\bar{\mathcal{A}}^2_{L(k)})_k.$$

That is, $(\bar{\mathcal{B}} \| \widehat{\mathcal{A}}^1)_k \leq^{(\hat{F})_k}_{p'(k),q(k),\epsilon'(k)} (\bar{\mathcal{B}} \| \widehat{\mathcal{A}}^2)_k$. This completes the proof. □

8 Application: Digital Timestamping

In this section, we present a formal model of the digital timestamping protocol of Haber et al. (cf. Section 1). Recall the real and ideal signature services from Section 6. The timestamping protocol consists of a dispatcher component and a collection of real signature services. Similarly, the ideal protocol consists of the same dispatcher with a collection of ideal signature services. Using the sequential composition theorem (Theorem 7.7), we prove that the real protocol implements the ideal protocol with respect to the long-term implementation relation $\leq^{\bar{F}}_{\mathsf{neg,pt}}$.

Let SID, the domain of service names, be $\mathbb{N}$. In addition to alive and $\mathsf{aliveTimes}$ (cf. Section 4), we assume the following.

- $\mathsf{pref} : \mathbb{T} \to SID$. For every $t \in \mathbb{T}$, the service $\mathsf{pref}(t)$ is the designated signer for time t, i.e., any signing request sent by the dispatcher at time t goes to service $\mathsf{pref}(t)$.
- $\mathsf{usable} : \mathbb{T} \to 2^{SID}$. For every $t \in \mathbb{T}$, $\mathsf{usable}(t)$ specifies the set of services that are accepting new verification requests.

Assume, for every $t \in \mathbb{T}$, $\mathsf{pref}(t) \in \mathsf{usable}(t) \subseteq \mathsf{alive}(t)$. If a service is preferred, it accepts both signing and verification requests. If it is alive but not usable, no new verification requests are accepted, but those already submitted will still be processed.

Dispatcher We define $\mathsf{Dispatcher}_k$ for each security parameter k. If the environment sends a first-time certificate request $\mathsf{reqCert}(rid, x)$, $\mathsf{Dispatcher}_k$ requests a signature from service $j = \mathsf{pref}(t)$ via the action $\mathsf{reqSign}(rid, \langle x, t, \bot \rangle)_j$, where t is the clock reading at the time of $\mathsf{reqSign}$. In this communication, we instantiate the message space M_k as $X_k \times \mathbb{T}_k \times (\Sigma_k)_\bot$, where X_k is the domain of documents to which timestamps are associated. After service j returns with action $\mathsf{respSign}(rid, \sigma)_j$, $\mathsf{Dispatcher}_k$ issues a new certificate via $\mathsf{respCert}(rid, \sigma, j)$.

If a renew request $\mathsf{reqCert}(rid, x, t, \sigma_1, \sigma_2, j)$ comes in, $\mathsf{Dispatcher}_k$ first checks to see if j is still usable. If not, it responds with $\mathsf{respCert}(rid, \mathsf{false})$. Otherwise, it sends $\mathsf{reqVer}(rid, \langle x, t, \sigma_1 \rangle, \sigma_2)_j$ to service j. If service j answers affirmatively, $\mathsf{Dispatcher}_j$ sends a signature request $\mathsf{reqSign}(rid, \langle x, t, \sigma_2 \rangle)_{j'}$, where j' is the current preferred service. When service j' returns with action $\mathsf{respSign}_{j'}(rid, \sigma_3)$, $\mathsf{Dispatcher}_k$ issues a new certificate via $\mathsf{respCert}(rid, \sigma_3, j')$.

The code for Dispatcher appears in The task-PIOA code for the component Dispatcher appears in Figure 6. As a convention, we use σ_1, σ_2 and σ_3 to denote previous, current, and new signatures, respectively.

Concrete Time Scheme Let d be a positive natural number. Each service j is alive from time $(j-1)\cdot d$ to $(j+2)\cdot d$. Thus, at any given point in time, there can be at most three services that are concurrently alive. Moreover, service j is preferred for signing from time $(j-1)\cdot d$ to $j\cdot d$, and is usable from time $(j-1)\cdot d$ to $(j+1)\cdot d$. Between $(j+1)\cdot d$ and $(j+2)\cdot d$, services j continues to process requests already submitted, without receiving new requests.

Protocol Correctness For every security parameter k, let $SID_k \subseteq SID$ denote the set of $p(k)$-bit numbers, for some polynomial p. Recall from Section 6 that $\mathsf{RealSig}(j)_k = \mathsf{hide}(\mathsf{KeyGen}(k,j)\|\mathsf{Signer}(k,j)\|\mathsf{Verifier}(k,j), \mathsf{signKey}_j)$ and $\mathsf{IdealSig}(j)_k = \mathsf{hide}(\mathsf{KeyGen}(k,j)\|\mathsf{SigFunc}(k,j), \mathsf{signKey}_j)$. Here we define

$$\mathsf{RealSigSys}_k := \mathsf{Dispatcher}_k\|(\|_{j\in SID_k}\mathsf{RealSig}(j)_k) \text{ and } \mathsf{IdealSigSys}_k := \mathsf{Dispatcher}_k\|(\|_{j\in SID_k}\mathsf{IdealSig}(j)_k).$$

Next, define $\overline{\mathsf{RealSigSys}} := \{\mathsf{RealSigSys}_k\}_{k\in\mathbb{N}}$ and $\overline{\mathsf{IdealSigSys}} := \{\mathsf{IdealSigSys}_k\}_{k\in\mathbb{N}}$. Our goal is to show that

$$\overline{\mathsf{RealSigSys}} \leq^{\bar{F}}_{\mathsf{neg,pt}} \overline{\mathsf{IdealSigSys}},$$

where $\bar{F}_k := \bigcup_{j\in SID_k}\{\{\mathsf{fail}_j\}\}$ for every k (Theorem 8.4).

First we make a key observation.

Lemma 8.1. *Suppose we have $k\in\mathbb{N}$, $j\in SID_k$, and $\mathcal{B}$ compatible with $\mathsf{RealSig}(j)_k$. Then $\mathsf{RealSig}(j)_k$ is restricted to $[(j-1)\cdot d, (j+2)\cdot d]$ by $\mathcal{B}$. Similarly for $\mathsf{IdealSig}(j)_k$.*

Proof. Suppose we have $t < (j-1)\cdot d$, environment Env for $\mathsf{RealSig}(j)_k\|\mathcal{B}$ of the form $\mathsf{Env}'\|\mathsf{Clock}$, valid schedule τ for $\mathsf{RealSig}(j)_k\|\mathcal{B}\|\mathsf{Env}$ for $[0,t]$, and state s reachable under τ. Recall from Section 3 that, for every $t'\in\mathbb{T}$, the action $\mathsf{tick}(t')$ must take place at time t'. Therefore, τ does not trigger a $\mathsf{tick}(t')$ action with $t'\in[(j-1)\cdot d, (j+2)\cdot d]$. On the other hand, all variables of $\mathsf{RealSig}(j)_k$ remains $\perp$ unless such a $\mathsf{tick}(t')$ action takes place, so we can conclude that $s.v = \perp$ for every variable v of $\mathsf{RealSig}(j)_k$.

For $t > (j+2)\cdot d$, we know that τ must have triggered the action $\mathsf{tick}((j+2)\cdot d)$, which sets all variables of $\mathsf{RealSig}(j)_k$ to $\perp$. Moreover, every subsequent $\mathsf{tick}(t')$ has $t' > t$, therefore the variables remain $\perp$.

Finally, by inspection of the code for $\mathsf{RealSig}(j)_k$, we know that no locally controlled actions are enabled if all variables are $\perp$.

The proof for $\mathsf{IdealSig}(j)_k$ is similar. □

For each $i\in\{0,1,2\}$, define $\mathsf{Real}_{i,k}$ to be the parallel composition of all $\mathsf{RealSig}(j)_k$ with $(j-1) \mod 3 = i$. Let $\overline{\mathsf{Real}_i}$ be $\{\mathsf{Real}_{i,k}\}_{k\in\mathbb{N}}$. By Lemma 8.1, we know that $\mathsf{RealSig}(i)_k, \mathsf{RealSig}(i+3)_k, \ldots$ are sequential under $\mathcal{B}$ for any $\mathcal{B}$. Thus, we have partitioned the collection of real signature services into three classes, $\overline{\mathsf{Real}_0}$, $\overline{\mathsf{Real}_1}$, and $\overline{\mathsf{Real}_2}$, such that the services within each $\overline{\mathsf{Real}_i}$ are sequential. For instance, the first class consists of services $0, 3, \ldots$, which are alive in intervals $[0, 3d], [3d, 6d], \ldots$ respectively.

Define $\mathsf{Ideal}_{i,k}$ and $\overline{\mathsf{Ideal}_i}$ similarly. We make the following observations.

Lemma 8.2. *The following families are polynomially bounded.*

1. $\overline{\mathsf{Dispatcher}}\|\overline{\mathsf{Real}_0}\|\overline{\mathsf{Real}_1}\|\overline{\mathsf{Real}_2}$.
2. $\overline{\mathsf{Dispatcher}}\|\overline{\mathsf{Ideal}_0}\|\overline{\mathsf{Real}_1}\|\overline{\mathsf{Real}_2}$.
3. $\overline{\mathsf{Dispatcher}}\|\overline{\mathsf{Ideal}_0}\|\overline{\mathsf{Ideal}_1}\|\overline{\mathsf{Real}_2}$.
4. $\overline{\mathsf{Dispatcher}}\|\overline{\mathsf{Ideal}_0}\|\overline{\mathsf{Ideal}_1}\|\overline{\mathsf{Ideal}_2}$.

Lemma 8.3. *The following hold for every k.*

1. $\mathsf{RealSig}(1)_k, \mathsf{RealSig}(4)_k, \ldots$ *in* Real_0 *and* $\mathsf{IdealSig}(1)_k, \mathsf{IdealSig}(4)_k, \ldots$ *in* Ideal_0 *are sequential under the automaton* $\overline{\mathsf{Dispatcher}}\|\overline{\mathsf{Real}_1}\|\overline{\mathsf{Real}_2}$ *for the sequence* $0 < 3d < 6d < \ldots$
2. $\mathsf{RealSig}(2)_k, \mathsf{RealSig}(5)_k, \ldots$ *in* Real_1 *and* $\mathsf{IdealSig}(j)_k, \mathsf{IdealSig}(5)_k, \ldots$ *in* Ideal_1 *are sequential under the automaton* $\overline{\mathsf{Dispatcher}}\|\overline{\mathsf{Ideal}_0}\|\overline{\mathsf{Real}_2}$ *for the sequence* $d < 4d < \ldots$
3. $\mathsf{RealSig}(3)_k, \mathsf{RealSig}(6)_k, \ldots$ *in* Real_2 *and* $\mathsf{IdealSig}(3)_k, \mathsf{IdealSig}(6)_k, \ldots$ *in* Ideal_2 *are sequential under the automaton* $\overline{\mathsf{Dispatcher}}\|\overline{\mathsf{Ideal}_0}\|\overline{\mathsf{Ideal}_2}$ *for the sequence* $2d < 5d < \ldots$

Proof. Follows directly from Lemma 8.1. □

Since each ideal service j has the same lifetime as the real service j, we can apply Theorem 7.7 to replace $\overline{\mathsf{Real}_i}$ with $\overline{\mathsf{Ideal}_i}$. This is the core step in the proof of the following correctness theorem.

Theorem 8.4. *Assume the concrete time scheme described above and that every signature scheme used in the timestamping protocol is complete and existentially unforgeable. By Theorem 6.2, this implies* $\overline{\mathsf{RealSig}}(j) \leq_{\mathsf{neg,pt}}^{\{\mathsf{fail}_j\}} \overline{\mathsf{IdealSig}}(j)$ *for every* $j \in SID$*. Assume further that, for every pair of polynomials* $\langle p, q\rangle$*, there exists a negligible function* ϵ *such that* $\overline{\mathsf{RealSig}}(j) \leq_{p,q,\epsilon}^{\{\mathsf{fail}_j\}} \overline{\mathsf{IdealSig}}(j)$ *for every* $j \in SID$*. Then* $\overline{\mathsf{RealSigSys}} \leq_{\mathsf{neg,pt}}^{\bar{F}} \overline{\mathsf{IdealSigSys}}$*, where* $\bar{F}_k := \bigcup_{j\in SID_k} \{\{\mathsf{fail}_j\}\}$ *for every* k*.*

Proof. We apply Theorem 7.7 three times:

1. Instantiate $\bar{B}$ with $\overline{\mathsf{Dispatcher}}\|\overline{\mathsf{Real}_1}\|\overline{\mathsf{Real}_2}$ and $\mathcal{A}$ with $\overline{\mathsf{Real}_0}$.
2. Instantiate $\bar{B}$ with $\overline{\mathsf{Dispatcher}}\|\overline{\mathsf{Ideal}_0}\|\overline{\mathsf{Real}_2}$ and $\mathcal{A}$ with $\overline{\mathsf{Real}_1}$, and
3. Instantiate $\bar{B}$ with $\overline{\mathsf{Dispatcher}}\|\overline{\mathsf{Ideal}_0}\|\overline{\mathsf{Ideal}_1}$ and $\mathcal{A}$ with $\overline{\mathsf{Real}_2}$.

Step 1: It is easy to see that for each and $j \in SID$, $\mathsf{RealSig}_j \in \mathsf{Real}_0$ is comparable to $\mathsf{IdealSig}_j \in \mathsf{Ideal}_0$. Observe also that compatibility conditions are also satisfied. The number of components in $\mathsf{Real}_{0,k}$ is bounded by the cardinality of the set SID_k. Since SID_k is the set of $p(k)$-bit numbers for some polynomial p, the size of SID_k is bounded by some exponential in k. We use this exponential for the L bound in Theorem 7.7. By Lemma 8.2 Parts 1 and 2, we know that conditions on the complexity bounds are met. By Lemma 8.3 Part 1, we exhibit the needed sequence of positive reals for sequentiality. By Theorem 6.2, we have for every pair polynomials p and q, there exists a negligible function such that $\mathsf{RealSig}_j \leq_{p,q,\epsilon}^{\{\{\mathsf{fail}_j\}\}} \mathsf{IdealSig}_j$.

By the result of Step 1, we get $(\mathsf{Dispatcher}\|\overline{\mathsf{Real}_0}\|\overline{\mathsf{Real}_1}\|\overline{\mathsf{Real}_2}) \leq_{\mathsf{neg,pt}}^{\bar{F}_0} (\mathsf{Dispatcher}\|\overline{\mathsf{Ideal}_0}\|\overline{\mathsf{Real}_1}\|\overline{\mathsf{Real}_2})$, where $(\bar{F}_0)_k = \{\{\mathsf{fail}_0\}, \{\mathsf{fail}_3\}, \ldots\}$ for every k.

Step 2: Similar to Step 1, using Part 2 and 3 of Lemma 8.2 and Part 2 of Lemma 8.3. By the result of step 2, we get $\mathsf{Dispatcher}\|\overline{\mathsf{Ideal}_0}\|\overline{\mathsf{Real}_1}\|\overline{\mathsf{Real}_2}$ implements $\mathsf{Dispatcher}\|\overline{\mathsf{Ideal}_0}\|\overline{\mathsf{Ideal}_1}\|\overline{\mathsf{Real}_2}$, where, for every k, $(\bar{F}_1)_k$ is the set $\{\{\mathsf{fail}_0\}, \{\mathsf{fail}_1\}, \{\mathsf{fail}_3\}, \{\mathsf{fail}_4\}, \ldots\}$.

Step 3: Similar to Step 2, using Part 3 and 4 of Lemma 8.2 and Part 3 of Lemma 8.3. By the result of step 3, we get $\mathsf{Dispatcher}\|\overline{\mathsf{Ideal}_0}\|\overline{\mathsf{Ideal}_1}\|\overline{\mathsf{Real}_2}$ implements $\mathsf{Dispatcher}\|\overline{\mathsf{Ideal}_0}\|\overline{\mathsf{Ideal}_1}\|\overline{\mathsf{Ideal}_2}$, where, for every k, $(\bar{F})_k$ is the set $\{\{\mathsf{fail}_0\}, \{\mathsf{fail}_1\}, \{\mathsf{fail}_2\}, \ldots\}$.

Finally, we combine these three using transitivity (Lemma 5.3).

□

9 Conclusion

We augment the Task-PIOA model with real time information on task schedules. This allows us to express computational restrictions in terms of processing rates with respect to real time. As demonstrated by the Turing machine simulation of Section 4, this new complexity model is similar to the standard PSPACE model.

The long-term implementation relation $\leq_{\mathsf{neg,pt}}$ is largely inspired by the timestamping service example of Section 8. We capture the idea that, while an unbounded environment will eventually succeed in guessing a secret key, we could control the rate at which these successes occur. By virtue of the sequential composition theorem, it is sufficient to analyze each signature service in isolation, checking that the adversary cannot break the service too quickly.

In the future, we plan to study general security definitions based on long-term implementation, and to conduct formal analysis of practical long-lived protocols. In addition, we plan to generalize our framework to allow the computational power of the various system components to increase with time.

References

[1] D. Bayer, S. Haber, and S. W. Stornetta. Improving the efficiency and reliability of digital time-stamping. In R. M. Capocalli, A. De Santis, , and U. Vaccaro, editors, *Sequences II: Methods in Communication, Security, and Computer Science*, pages 329–334. Springer-Verlag, 1993. (Proceedings of the Sequences Workshop, 1991).

[2] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In Moni Naor, editor, *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, pages 136–145. IEEE Computer Society, 2001. Full version available on `http://eprint.iacr.org/2000/067`.

[3] Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Analyzing security protocols using time-bounded Task-PIOAs. *Discrete Event Dynamic Systems*, 18(1), 2008. 49 p., to appear. (Full version available on `http://eprint.iacr.org/2005/452`).

[4] S. Haber. Long-lived digital integrity using short-lived hash functions. Technical report, HP Laboratories, May 2006.

[5] S. Haber and P. Kamat. A content integrity service for long-term digital archives. In *Proceedings of the IS&T Archiving Conference*, 2006. Also published as Technical Memo HPL-2006-54, Trusted Systems Laboratory, HP Laboratories, Princeton.

[6] R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006)*, pages 309–320. IEEE Computer Society, 2006.

[7] P.D. Lincoln, J.C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of ACM CCS-5*, 1998.

[8] N.A. Lynch and M.R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, September 1989.

[9] M. Merritt, F. Modugno, and M.R. Tuttle. Time constrained automata. In *Proceedings of CONCUR 1991*, volume 527 of *LNCS*, pages 408–423, 1991.

[10] J. Müller-Quade and D. Unruh. Long-term security and universal composability. In *Theory of Cryptography, Proceedings of TCC 2007*, volume 4392 of *LNCS*, pages 41–60. Springer-Verlag, March 2007. Preprint on IACR ePrint 2006/422.

[11] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–200, Oakland, CA, May 2001. IEEE Computer Society.

Dispatcher$(k : \mathbb{N})$

Signature

```
Input:
    tick($t : \mathbb{T}_k$)
    reqCert($rid : RID_k, x : X_k$)
    reqCert($rid : RID_k, x : X_k, t : \mathbb{T}_k,$
       $\sigma_1 : (\Sigma_k)_\perp, \sigma_2 : \Sigma_k, j : SID$)
    reqCheck($rid : RID_k, x : X_k, t : \mathbb{T}_k,$
       $\sigma_1 : (\Sigma_k)_\perp, \sigma_2 : \Sigma_k, j : SID$)
    respSign$(rid : RID_k, \sigma : \Sigma_k)_j, j \in SID$
    respVer$(rid : RID_k, b : Bool)_j, j \in SID$
Output:
    reqSign$(rid : RID_k, m : M_k)_j, j \in SID$
    reqVer$(rid : RID_k, m : M_k, \sigma : \Sigma_k)_j, j \in SID$
    respCert($rid : RID_k, \sigma : \Sigma_k, j : SID$)
    respCert($rid : RID_k$, false)
    respCheck($rid : RID_k, b : Bool$)
Internal:
    denyVer($rid : RID_k, op : \{'cert', 'check'\},$
       $m : M_k, \sigma : \Sigma_k, j : SID$)
```

Tasks

```
reqSign $= \{$reqSign$(*,*)_*\}$
reqVer $= \{$reqVer$(*,*,*)_*\}$
respCert $= \{$respCert$(*,*,*)\} \cup \{$respCert$(*,$ false$)\}$
respCheck $= \{$respCheck$(*,*)\}$
denyVer $= \{$denyVer$(*,*,*,*,*)\}$
```

States

```
$clock : \mathbb{T}_k$, init 0
$toSign$ : que$(RID_k \times M)$, init empty
$toVer$ : que$(RID_k \times \{'cert', 'check'\}$
$\times M \times \Sigma \times SID)$, init empty
$pendingVer, pendingSign : Bool$, init false
$certified$ : que$((RID_k \times \Sigma \times SID)$
$\cup (RID_k \times \{false\}))$, init empty
$checked$ : que$(RID_k \times Bool)$, init empty
$currCt : \mathbb{N}$, init 0
```

Transitions

```
tick($t$)
Effect:
    $clock := t$

reqCert($rid, x$)
Effect:
    if $currCt < b$ then
       $toSign :=$ enq$(toSign, \langle rid, \langle x, clock, \perp \rangle\rangle)$
       $currCt := currCt + 1$

reqCert($rid, x, t, \sigma_1, \sigma_2, j$)
Effect:
    if $currCt < b$ then
       $toVer :=$ enq$(toVer, \langle rid, 'cert', \langle x, t, \sigma_1 \rangle, \sigma_2, j\rangle)$
       $currCt := currCt + 1$

reqCheck($rid, x, t, \sigma_1, \sigma_2, j$)
Effect:
    if $currCt < b$ then
       $toVer :=$ enq$(toVer, \langle rid, 'check', \langle x, t, \sigma_1 \rangle, \sigma_2, j\rangle)$
       $currCt := currCt + 1$

reqSign$(rid, m)_j$
Precondition:
    head$(toSign) = \langle rid, m \rangle$
    $j =$ pref$(clock)$
    $\neg pendingSign$
Effect:
    $pendingSign := true$

respSign$(rid, \sigma_3)_j$
Effect:
    if $pendingSign \wedge (\exists m)($head$(toSign) = \langle rid, m, j\rangle)$ then
       choose $m$ where head$(toSign) = \langle rid, m, j \rangle$
       $toSign :=$ deq$(toSign)$
       $pendingSign := false$
       choose $x, t$ where $(\exists \sigma_2)(m = \langle x, t, \sigma_2 \rangle)$
       $certified :=$ enq$(certified, \langle rid, \sigma_3, j\rangle)$

denyVer($rid, op, m, \sigma_2, j$)
Precondition:
    $head(toVer) = \langle rid, op, m, \sigma_2, j \rangle$
    $j \notin$ usable$(clock)$
Effect:
    $toVer :=$ deq$(toVer)$
    if $op = 'cert'$ then
       $certified :=$ enq$(certified, \langle rid, false \rangle)$
    else $checked :=$ enq$(checked, \langle rid, false \rangle)$

reqVer$(rid, m, \sigma_2)_j$
Precondition:
    $(\exists op)($head$(toVer) = \langle rid, op, m, \sigma_2, j \rangle$
    $j \in$ usable$(clock)$
    $\neg pendingVer$
Effect:
    $pendingVer := true$

respVer$(rid, b)_j$
Effect:
    if $pendingVer$
       $\wedge (\exists op, m, \sigma_2)(head(toVer) = \langle rid, op, m, \sigma_2, j \rangle)$ then
       choose $op, m, \sigma_2$ where head$(toVer) = \langle rid, op, m, \sigma_2, j \rangle$
       $toVer :=$ deq$(toVer)$
       $pendingVer := false$
       if $op = 'cert' \wedge \neg b$ then
          $certified :=$ enq$(certified, \langle rid, false \rangle)$
       if $op = 'cert' \wedge b$ then
          choose $x, t$ where $(\exists \sigma_1)(m = \langle x, t, \sigma_1 \rangle)$
          $toSign :=$ enq$(toSign, \langle rid, \langle x, t, \sigma_2 \rangle\rangle)$
       if $op = 'check'$ then
          $checked :=$ enq$(checked, \langle rid, b \rangle)$

respCert($rid, false$)
Precondition:
    head$(certified) = \langle rid, false \rangle$
Effect:
    $certified :=$ deq$(certified)$
    $currCt := currCt - 1$

respCert($rid, \sigma_3, j$)
Precondition:
    $head(certified) = \langle rid, \sigma_3, j \rangle$
Effect:
    $certified :=$ deq$(certified)$
    $currCt := currCt - 1$

respCheck($rid, b$)
Precondition:
    head$(checked) = \langle rid, b \rangle$
Effect:
    $checked :=$ deq$(checked)$
    $currCt := currCt - 1$
```

Figure 6: Task-PIOA Code for Dispatcher$(k : \mathbb{N})$