Modeling Computational Security in Long-Lived Systems *

Ran Canetti[1,2], Ling Cheung[2], Dilsun Kaynar[3], Nancy Lynch[2], and Olivier Pereira[4]

[1] IBM T. J. Watson Research Center
[2] Massachusetts Institute of Technology
[3] Carnegie Mellon University
[4] Université catholique de Louvain

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 a protocol. However, some cryptographic services, such as timestamping 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: a computationally secure protocol may become insecure if the attacker has a super-polynomial number of interactions with the protocol.
This paper 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 of real time*. Moreover, the space used by these entities is allocated dynamically and must be polynomially bounded. We propose a new 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. We illustrate the use of this new paradigm by analyzing some security properties of the long-lived timestamping protocol of Haber and Kamat.

1 Introduction

Computational security in long-lived systems: Security properties of cryptographic protocols typically hold only against resource-bounded adversaries. Consequently, mathematical models for representing and analyzing security of such protocols usually represent all participants as resource-bounded computational entities. The predominant way of formalizing such bounds is by representing

* Canetti's work on this project was supported by NSF award #CFF-0635297 and BSF award #2006317. Cheung and Lynch were supported by NSF Award #CCR-0326227. Kaynar was supported by US Army Research Office grant #DAAD19-01-1-0485. Pereira is a Research Associate of the F.R.S.-FNRS and was supported by the Belgian Interuniversity Attraction Pole P6/26 BCRYPT.

all entities as time-bounded machines, specifically, polynomial-time machines (a partial list of works representative of this direction includes [1–5]).

This modeling approach has been successful in capturing the security of protocols for many cryptographic tasks. However, it has a fundamental limitation: it assumes that the analyzed system runs for only a relatively "short" time. In particular, since all entities are polynomially-bounded (in the security parameter), the system's execution must end after a polynomial amount of time. This type of modeling is inadequate for analyzing security properties of protocols that are supposed to run for a "long" time, that is, an amount of time that is not bounded by a polynomial.

There are a number of natural tasks for which one would indeed be interested in the behavior of systems that run for a long time. Furthermore, a number of protocols have been developed for such tasks. Examples of such tasks include proactive security [6], forward secure signatures [7, 8], forward secure encryption [7, 9], and timestamping [10–12]. None of the existing models for analyzing security against computationally bounded adversaries is adequate for asserting and proving security properties of protocols for such "long-lived" tasks.

Related work: A first suggestion for an approach might be to use existing models, such as the PPT calculus [13], the Reactive Simulatability [14], or the Universally Composable security frameworks [3], with a sufficiently large value of the security parameter. However, this would be too limited for our purpose in that it would force protocols to protect against an overly powerful adversary *even in the short run,* while not providing any useful information in the long run. Similarly, turning to information theoretic security notions is not appropriate in our case because unbounded adversaries would be able to break computationally secure schemes instantaneously. We are interested in a notion of security that can protect protocols against an adversary that runs for a long time, but is only "reasonably powerful" at any point in time.

Recently, Müller-Quade and Unruh proposed a notion of *long-term security* for cryptographic protocols [15]. However, they consider adversaries that try to derive information from the protocol transcript *after* protocol conclusion. This work does not consider long-lived protocol execution and, in particular, the adversary of [15] has polynomially bounded interactions with the protocol parties, which is not suitable for the analysis of long-lived tasks such as those we described above.

Our approach: In this paper, we propose a new mathematical model for analyzing the security of such *long-lived systems.* To the best of our knowledge our work is the first one to tackle the issue of modeling computational security in long-lived systems. Our understanding of a long-lived system is that some protocol parties, including adversaries, 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 of real time. Other parties may be active for only a short time, as in traditional settings. Thus, the adversary's interaction with the system is unbounded, and the adversary may perform an unbounded number of computation steps during the entire protocol execution. This renders traditional security

notions insufficient: computationally and even statistically secure protocols may fail if the adversary has unbounded interactions with the protocol.

Modeling long-lived systems requires significant 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 very rapidly in a short amount of time. The former conforms with our understanding of boundedness, while the latter does not. Guided by this intuition, we introduce real time explicitly into a basic probabilistic automata model, the Task-PIOA model [5], and impose computational restrictions in terms of *rates*, i.e., number of computation steps per unit of real time.

Another interesting challenge is the restriction on space, which traditionally is not an issue because PPT Turing machines can, by their nature, access only a polynomially bounded amount of space. In the long-lived setting, space restriction warrants explicit consideration. During the lifetime of a long-lived security protocol, we expect some components to die and other new ones to become active, for example, due to the use of cryptographic primitives that have a shorter life time than the protocol itself. Therefore, we find it important to be able to model dynamic allocation of space. We achieve this by restricting the use of state variables. In particular, all state variables of a dormant entity (either not yet invoked or already dead) are set to a special null 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 then define a new *long-term implementation relation*, $\leq_{\mathsf{neg,pt}}$, for long-lived systems. This is intended to extend 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. However, 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 is true even if the environment has bounded computation rate. Therefore, our definition cannot 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 during any polynomial-bounded window of time remains bounded during the lifetime of the system.

To capture this intuition, we extend the ideal systems traditionally used in cryptography by allowing them to take some designated *failure* steps, which allow an ideal system to take actions that could only occur in the real world, e.g., accepting forgeries as valid signatures, or producing ciphertexts that could allow recovering the corresponding plaintext. However, if failure steps do not occur

starting from some time t, then the ideal system starts following the specified ideal behavior.

Our long-term implementation relation $\leq_{\mathsf{neg,pt}}$ requires that the real system approximates the ideal's system's handling of failures. More precisely, we quantify over all real time points t and require that the real and ideal systems are computationally indistinguishable up to time $t+q$ (where q is polynomial in the security parameter), even if no failures steps are taken by the ideal system in the interval $[t, t+q]$. 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. Our formal definition of $\leq_{\mathsf{neg,pt}}$ includes one more generalization: it considers failure steps in the real system as well as the ideal system, in both cases before the same real time t. This natural extension is intended to allow repeated use of $\leq_{\mathsf{neg,pt}}$, in verifying protocols using several levels of abstraction.

We show that $\leq_{\mathsf{neg,pt}}$ is transitive, and is preserved under the operations of polynomial parallel composition and exponential sequential composition. The sequential composition result highlights the power of our model to formulate and prove properties of an exponential number of entities in a meaningful way.

Example: Digital timestamping: As a proof of concept, we analyze some security properties of the digital timestamping protocol of Haber et al. [10–12], 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 amount 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.

2 Task-PIOAs

We build our new framework using task-PIOAs [5], which are a version of Probabilistic Automata [16], augmented with an oblivious 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 groupings of events, both for real time scheduling and for imposing computational bounds (cf. Sections 3 and 4). In this section, we review basic notations related to task-PIOAs.

Notation: 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, for any set S with $\perp \notin S$, we write $S_\perp := S \cup \{\perp\}$.

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 $O \cup H$. An *execution* is a sequence $\alpha = q_0 a_1 q_1 a_2 \ldots$ of alternating states and actions where $q_0 = s^{\mathsf{init}}$ and, for each $\langle q_i, a_{i+1}, q_{i+1} \rangle$, there is a transition $\langle q_i, a_{i+1}, \mu \rangle \in \Delta$ with $q_{i+1} \in \mathsf{Support}(\mu)$. A sequence obtained by restricting an execution of $\mathcal{A}$ to 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 $\mathcal{A}$ satisfy 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-PIOA: To resolve nondeterminism, we make use of the notion of tasks introduced in [17, 5]. 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 additional 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 that $\mathbb{T} = \mathbb{N}$, and write simply Clock. Clock has 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 will ensure that each $\mathsf{tick}(t)$ occurs exactly at real time t.

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

Signature	**Tasks:** $\mathsf{tick} = \{\mathsf{tick}(*)\}$
Output: $\mathsf{tick}(t : \mathbb{T})$, $t > 0$	**States:** $count \in \mathbb{T}$, initially 0
Transitions	
$\mathsf{tick}(t)$	
Precondition: $count = t - 1$	Effect: $count := t$

Fig. 1. Task-PIOA Code for $\mathsf{Clock}(\mathbb{T})$

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.

We also define a *hiding* operator: given $\mathcal{A} = \langle V, S, s^{\mathsf{init}}, I, O, H, \Delta \rangle$ and $B \subseteq O$, $\mathsf{hide}(\mathcal{A}, B)$ is the task-PIOA given by $\langle V, S, s^{\mathsf{init}}, I, O', H', \Delta \rangle$, where $O' = O \setminus B$ and $H' = H \cup B$. This prevents other PIOAs from synchronizing with $\mathcal{A}$ via actions in B: any PIOA with an action in B 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 executions in $\mathcal{A}$. For finite ρ, let $\mathsf{lstate}(\mathcal{A}, \rho)$ denote the state

distribution of $\mathcal{A}$ after executing according to ρ. A state s is said to be *reachable* under ρ if $\mathsf{lstate}(\mathcal{A}, \rho)(s) > 0$. Moreover, the probabilistic execution induces a unique *trace distribution* $\mathsf{tdist}(\mathcal{A}, \rho)$, which is a probability distribution over the set of traces of $\mathcal{A}$. We refer the reader to [5] for more details on these constructions.

3 Real Time Scheduling Constraints

In this section, we describe how 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. Thus, we define a *timed* task schedule τ for a closed task-PIOA $\langle \mathcal{A}, \mathcal{R} \rangle$ to be 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. Given a timed task schedule $\tau = \langle T_1, t_1 \rangle, \langle T_2, t_2 \rangle, \ldots$ and $t \in \mathbb{R}_{\geq 0}$, let $\mathsf{trunc}_{\geq t}(\tau)$ denote the result of removing all pairs $\langle T_i, t_i \rangle$ with $t_i \geq t$.

Following [18], 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) $\mathsf{ub} : \mathcal{R} \to \mathbb{R}^{\infty}_{\geq 0}$, 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.

In the long version of this paper [19, Section 3], we formally define what it means for a timed task schedule τ to be valid for an interval under a given bound map. This definition states the technical conditions that simultaneously ensure that: (i) Consecutive appearances of a task T must be at least $\mathsf{lb}(T)$ apart, (ii) Consecutive appearances of a task T must be at most $\mathsf{ub}(T)$ apart, (iii) For any $d \in \mathbb{R}_{\geq 0}$ and any interval I' of length d, τ contains at most $\mathsf{rate}(T) \cdot d + \mathsf{burst}(T)$ elements with $\langle T, t \rangle$ 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 $\mathsf{tdist}(\mathcal{A}, \tau) := \mathsf{tdist}(\mathcal{A}, \rho)$.

In a parallel composition $\mathcal{A}_1 \| \mathcal{A}_2$, the composite bound map is the union of component bound maps: $\langle \mathsf{rate}_1 \cup \mathsf{rate}_2, \mathsf{burst}_1 \cup \mathsf{burst}_2, \mathsf{lb}_1 \cup \mathsf{lb}_2, \mathsf{ub}_1 \cup \mathsf{ub}_2 \rangle$.

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.

Note that we use real time solely to express constraints on task schedules. We do not allow computationally-bounded system components to maintain real-time information in their states, nor to communicate real-time information to each other. System components that require knowledge of time will maintain discrete approximations to time in their states, based on inputs from Clock.

4 Complexity Bounds

We are interested in modeling systems that run for an unbounded amount of real time. During this long life, we expect that a very large number of components will be active at various points in time, while only a small proportion of them will be active simultaneously. Defining complexity bounds in terms of the total number of components would then introduce unrealistic security constraints. Therefore, we find it more reasonable to define complexity bounds in terms of the characteristics of the components that are simultaneously active at any point in time.

To capture these intuitions, we define a notion of *step bound*, which limits the amount of computation a task-PIOA can perform, and the amount of space it can use, in executing a single step. By combining the step bound with the rate and burst bounds of Section 3, we obtain an *overall bound*, encompassing both bounded memory and bounded computation rates.

Note that we do not model situations where the rates of computation, or the computational power of machines, increases over time. This is an interesting direction in which the current research could be extended.

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 $\langle v, \perp\rangle$. 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 (a) the value of every variable is representable by at most p bits, (b) the name of every action name has length at most p bits, (c) the initial state s^{init} is p-bounded, (d) there are probabilistic Turing machines able to (i) determine which tasks are enabled in a given state of $\mathcal{A}$, (ii) determine which action a of a given task is enabled in a given state s of $\mathcal{A}$, and output a new state of $\mathcal{A}$ according to the distribution of $\Delta(s, a)$, (iii) determine if a candidate action a is an input action of $\mathcal{A}$ and, given a state s of $\mathcal{A}$, output a new state of $\mathcal{A}$ according to the distribution of $\Delta(s, a)$. Furthermore, those Turing Machines terminate after at most p steps on every input and they can be encoded using at most p bits.

Given a closed (i.e., no input actions) task-PIOA $\mathcal{A}$ with step bound p, one can easily define a Turing machine $M_{\mathcal{A}}$ with a combination of nondeterministic and probabilistic branching that simulates the execution of $\mathcal{A}$. It can be showed that the amount of work tape needed by $M_{\mathcal{A}}$ is polynomial in p.

It can also be shown 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 [5, Lemma 4.2]. We also note that the hiding operator introduced in Section 2 preserves step bounds.

Overall Bound: We now combine 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 $s \in S_t$. (Here, $s.count$ is the value of variable *count* of Clock in state s).

We say that $\mathcal{A}$ is *quasi-p-bounded* if $\mathcal{A}$ is of the form $\mathcal{A}' \| \mathsf{Clock}$ where $\mathcal{A}'$ is p-bounded.

Conditions (i) and (ii) are self-explanatory. Condition (iii) is a technical condition that ensures that the enabling of tasks does not change too rapidly. Without such a restriction, $\mathcal{A}$ could 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 now present an example of a polynomially bounded family of task-PIOAs—a signature service that we use in our digital timestamping example. The complete formal specification for these task-PIOAs can be found in the long version of this paper [19].

Example 3 (Signature Service). 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 $\langle k, j \rangle$, representing the key generator, signer, and verifier, respectively. The composition of these three task-PIOAs gives a signature service. 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 discrete time t. The lifetime of each service j is then given by $\mathsf{aliveTimes}(j) := \{t \in \mathbb{T} | j \in \mathsf{alive}(t)\}$; we assume this to be a finite set of consecutive numbers.

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 $\mathsf{acc} = \{\mathsf{acc}\}$. 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 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 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 [11, 12] described in Section 1. After running for a long period of real time, a distinguisher environment may be able to forge a signature with non-negligible probability. As a result, it can distinguish the real system from an ideal timestamping system, in the traditional sense. However, the essence of the protocol is that such failures can in fact be tolerated, because they do not help the environment to forge *new* signatures, after a new, uncompromised signature service becomes active.

This timestamping example suggests that we need a new notion of long-term implementation that makes meaningful security guarantees in any polynomial-bounded window of time, in spite of past security failures. Our new implementation relation aims to capture this intuition.

First we define a comparability condition for task-PIOAs: $\mathcal{A}^1$ 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. To model security failure events in both automata, we let F^1 be a set of designated *failure tasks* of $\mathcal{A}^1$, and let F^2 be a set of *failure tasks* of A^2. We assume that each task in F^1 and F^2 has ∞ as its upper bound.

Given $t \in \mathbb{R}_{\geq 0}$ and an environment Env for both $\mathcal{A}^1$ and $\mathcal{A}^2$, we consider two experiments. In the first experiment, Env interacts with $\mathcal{A}^1$ according to some valid task schedule τ_1 of $\mathcal{A}^1 \| \mathsf{Env}$, where τ_1 does not contain any tasks from F^1 from time t onwards. 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^2 from time t onwards. Our definition requires that the first experiment "approximates" the second one, that is, if $\mathcal{A}^1$ acts ideally (does not perform any of the failure tasks in F^1) after time t, then it simulates $\mathcal{A}^2$, also acting ideally from time t onwards.

More specifically, we require that, for any valid τ_1, there exists a valid τ_2 as above such that the two executions are identical before time t from the point of view of the environment. That is, the probabilistic execution is the same before time t. Moreover, the two executions are overall *computationally indistinguishable*, namely, the difference in acceptance probabilities in these two experiments is negligible provided Env is computationally bounded.

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 $\mathsf{Execs}_{\mathcal{B}}(\mathcal{A} \| \mathcal{B}, \tau)$ denote the distribution of executions of $\mathcal{B}$ when executed with $\mathcal{A}$ under schedule τ.

Definition 1. *Let $\mathcal{A}^1$ and $\mathcal{A}^2$ be comparable task-PIOAs that are both compatible with* Clock. *Let F^1 and F^2 be sets of tasks of, respectively, $\mathcal{A}^1$ and $\mathcal{A}^2$, such that for any $T \in (F^1 \cup F^2)$, $\mathsf{ub}(T) = \infty$. Let $p, q \in \mathbb{N}$ and $\epsilon \in \mathbb{R}_{\geq 0}$ be given. Then we say that $(\mathcal{A}^1, F^1) \leq_{p,q,\epsilon} (\mathcal{A}^2, F^2)$ provided that the following is true: For every $t \in \mathbb{R}_{\geq 0}$, every quasi-p-bounded environment* Env, *and every valid timed schedule τ_1 for $\mathcal{A}^1 \| \mathsf{Env}$ for the interval $[0, t+q]$ that does not contain any pairs of the form $\langle T_i, t_i \rangle$ where $T_i \in F^1$ and $t_i \geq t$, there exists a valid timed schedule τ_2 for $\mathcal{A}^2 \| \mathsf{Env}$ for the interval $[0, t+q]$ such that:*

(i) $\mathsf{proj}_{\mathsf{Env}}(\tau_1) = \mathsf{proj}_{\mathsf{Env}}(\tau_2)$;
(ii) *τ_2 does not contain any pairs of the form $\langle T_i, t_i \rangle$ where $T_i \in F^2$ and $t_i \geq t$;*
(iii) $\mathsf{Execs}_{\mathsf{Env}}(\mathcal{A}^1 \| \mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_1)) = \mathsf{Execs}_{\mathsf{Env}}(\mathcal{A}^2 \| \mathsf{Env}, \mathsf{trunc}_{\geq t}(\tau_2))$;
(iv) $|\mathbf{P}_{\mathsf{acc}}(\mathcal{A}^1 \| \mathsf{Env}, \tau_1) - \mathbf{P}_{\mathsf{acc}}(\mathcal{A}^2 \| \mathsf{Env}, \tau_2)| \leq \epsilon$.

It can be observed that the $\leq_{p,q,\epsilon}$ is transitive up to additive errors [19].

The relation $\leq_{p,q,\epsilon}$ 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 $\bar{F}^1$ be a family of sets such that each $(\bar{F}^1)_k$ is a set of tasks of $(\bar{\mathcal{A}}^1)_k$ and let $\bar{F}^2$ be a family of sets such that each $(\bar{F}^2)_k$ is a set of tasks of $(\bar{\mathcal{A}}^2)_k$, satisfying the condition that each task of those sets has an infinite upper bound.

Let $\epsilon : \mathbb{N} \to \mathbb{R}_{\geq 0}$ and $p, q : \mathbb{N} \to \mathbb{N}$ be given. We say that $(\bar{\mathcal{A}}^1, \bar{F}^1) \leq_{p,q,\epsilon} (\bar{\mathcal{A}}^2, \bar{F}^2)$ just in case $((\bar{\mathcal{A}}^1)_k, (\bar{F}^1)_k) \leq_{p(k),q(k),\epsilon(k)} ((\bar{\mathcal{A}}^2)_k, (\bar{F}^2)_k)$ for every k.

Restricting our attention to negligible error and polynomial time bounds, we obtain the long-term implementation relation $\leq_{\mathsf{neg,pt}}$. 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 families $\bar{F}^1$ and $\bar{F}^2$, respectively, of $\bar{\mathcal{A}}^1$ and $\bar{\mathcal{A}}^2$, we say that $(\bar{\mathcal{A}}^1, \bar{F}^1) \leq_{\mathsf{neg,pt}} (\bar{\mathcal{A}}^2, \bar{F}^2)$ if $\forall p, q\ \exists \epsilon : (\bar{\mathcal{A}}^1, \bar{F}^1) \leq_{p,q,\epsilon} (\bar{\mathcal{A}}^2, \bar{F}^2)$, where p, q are polynomials and ϵ is a negligible function.

Example 4 (Ideal Signature Functionality). In order to illustrate the use of the relation $\leq_{\mathsf{neg,pt}}$ in our example, we specify an *ideal signature functionality* $\mathsf{SigFunc}$, and show that it is implemented by the real signature service of Section 4.

As with KeyGen, Signer, and $\mathsf{Verifier}$, each instance of $\mathsf{SigFunc}$ is parameterized with a security parameter k and an identifier j. 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* = 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* = true, then $\mathsf{SigFunc}(k, j)$ bypasses the check on *history*, so that its answers are identical to those from the real signature service.

Let us define $\mathsf{RealSig}(j)_k = \mathsf{hide}(\mathsf{KeyGen}(k, j) \| \mathsf{Signer}(k, j) \| \mathsf{Verifier}(k, j), \mathsf{signKey}_j)$ and $\mathsf{IdealSig}(j)_k = \mathsf{hide}(\mathsf{KeyGen}(k, j) \| \mathsf{SigFunc}(k, j), \mathsf{signKey}_j)$. We define families from those automata in the obvious way: $\overline{\mathsf{RealSig}} := \{\mathsf{RealSig}_k\}_{k \in \mathbb{N}}$ and $\overline{\mathsf{IdealSig}} := \{\mathsf{IdealSig}_k\}_{k \in \mathbb{N}}$. We show that the real signature service implements the ideal signature functionality. The proof, which relies on a reduction to standard properties of a signature scheme, can be found in [19].

Theorem 1. *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 $(\overline{\mathsf{RealSig}}(j), \{\}) \leq_{\mathsf{neg,pt}} (\overline{\mathsf{IdealSig}}(j), \{\mathsf{fail}_j\})$.*

6 Composition Theorems

In practice, cryptographic services are seldom used in isolation. Usually, 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 begins. An example of such a collection is the signature services in the timestamping protocol of [12, 11], 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.[5] This highlights a unique aspect of our implementation relation: essentially, from any point t on the real time line, we focus on a polynomial length interval starting from t.

Parallel Composition: Using a standard hybrid argument, as exemplified in [20] for instance, it is possible to show that the relation $\leq_{\mathsf{neg,pt}}$ is preserved under polynomial parallel composition. The theorem contains a technicality: instead of simply assuming $\leq_{\mathsf{neg,pt}}$ relationships for all the components, we assume a slightly stronger property, in which the same negligible function ϵ is assumed for all of the components; that is, ϵ is not allowed to depend on the component index i.

Theorem 2 (Parallel Composition Theorem for $\leq_{\mathsf{neg,pt}}$). *Let $\bar{\mathcal{A}}_1^1, \bar{\mathcal{A}}_2^1, \ldots$ and $\bar{\mathcal{A}}_1^2, \bar{\mathcal{A}}_2^2, \ldots$ be two infinite sequences of task-PIOA families, with $\bar{\mathcal{A}}_i^1$ comparable to $\bar{\mathcal{A}}_i^2$ for every i. Suppose that $\bar{\mathcal{A}}_1^{\alpha_1}, \bar{\mathcal{A}}_2^{\alpha_2}, \ldots$ are pairwise compatible for any combination of $\alpha_i \in \{1, 2\}$. Let b be any polynomial, and for each k, let $(\widehat{\mathcal{A}}^1)_k$ and $(\widehat{\mathcal{A}}^2)_k$ denote $\|_{i=1}^{b(k)}(\bar{\mathcal{A}}_i^1)_k$ and $\|_{i=1}^{b(k)}(\bar{\mathcal{A}}_i^2)_k$, respectively. Let r and s be polynomials, $r, s : \mathbb{N} \to \mathbb{N}$, such that r is nondecreasing, and for every i, k, both $(\bar{\mathcal{A}}_i^1)_k$ and $(\bar{\mathcal{A}}_i^2)_k$ are bounded by $s(k) \cdot r(i)$.*

For each i, let $\bar{F}_i^1$ be a family of sets such that $(\bar{F}_i^1)_k$ is a set of tasks of $(\bar{\mathcal{A}}_i^1)_k$ for every k, and let $\bar{F}_i^2$ be a family of sets such that $(\bar{F}_i^2)_k$ is a set of tasks of $(\bar{\mathcal{A}}_i^2)_k$ for every k, where all these tasks have infinite upper bounds. Let $(\hat{F}^1)_k$ and $(\hat{F}^2)_k$ denote $\bigcup_{i=1}^{b(k)} (\bar{F}_i^1)_k$ and $\bigcup_{i=1}^{b(k)} (\bar{F}_i^2)_k$, respectively.

Assume:

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

where p, q are polynomials and ϵ is a negligible function.

Then $(\widehat{\mathcal{A}}^1, \hat{F}^1) \leq_{\mathsf{neg,pt}} (\widehat{\mathcal{A}}^2, \hat{F}^2)$.

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

[5] 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.

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.[6] In Definition 2, we formalize the idea of an entity $\mathcal{A}$ being active during a particular time interval. Then we introduce sequentiality in Definition 3.

Definition 2. *Let $\mathcal{A}$ be a task-PIOA and let reals $t_1 \leq t_2$ be given. We say that $\mathcal{A}$ is* restricted to *the interval $[t_1, t_2]$ if for every $t \notin [t_1, t_2]$, environment Env for $\mathcal{A}$ of the form $\mathsf{Env}' \| \mathsf{Clock}$, valid schedule τ for $\mathcal{A} \| \mathsf{Env}$ for $[0, t]$, and state s reachable under τ, no locally controlled actions of $\mathcal{A}$ are enabled in s, and $s.v = \perp$ for every variable v of $\mathcal{A}$.*

Definition 3 (Sequentiality). *Let $\mathcal{A}_1, \mathcal{A}_2, \ldots$ be pairwise compatible task-PIOAs. We say that $\mathcal{A}_1, \mathcal{A}_2, \ldots$ are* sequential with respect to *the the nondecreasing sequence $t_1, t_2, \ldots$ of nonnegative reals provided that for every i, $\mathcal{A}_i$ is restricted to $[t_i, t_{i+1}]$.*

Note the slight technicality that each $\mathcal{A}_i$ may overlap with $\mathcal{A}_{i+1}$ at the boundary time t_{i+1}.

Theorem 3 (Sequential Composition Theorem for $\leq_{\mathsf{neg,pt}}$). *Let $\bar{\mathcal{A}}^1_1, \bar{\mathcal{A}}^1_2, \ldots$ and $\bar{\mathcal{A}}^2_1, \bar{\mathcal{A}}^2_2, \ldots$ be two infinite sequences of task-PIOA families, with $\bar{\mathcal{A}}^1_i$ comparable to $\bar{\mathcal{A}}^2_i$ for every i. Suppose that $\bar{\mathcal{A}}^{\alpha_1}_1, \bar{\mathcal{A}}^{\alpha_2}_2, \ldots$ are pairwise compatible for any combination of $\alpha_i \in \{1, 2\}$. Let $L : \mathbb{N} \to \mathbb{N}$ be an exponential function and, for each k, let $(\widehat{\mathcal{A}}^1)_k$ and $(\widehat{\mathcal{A}}^2)_k$ denote $\|_{i=1}^{L(k)} (\bar{\mathcal{A}}^1_i)_k$ and $\|_{i=1}^{L(k)} (\bar{\mathcal{A}}^2_i)_k$, respectively. Let $\hat{p}$ be a polynomial such that both $\widehat{\mathcal{A}}^1$ and $\widehat{\mathcal{A}}^2$ are $\hat{p}$-bounded.*

Suppose there exists an increasing sequence of nonnegative 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 for $t_1, t_2, \ldots$ Assume there is a constant real number c such that consecutive t_i's are at least c apart.

For each i, let $\bar{F}^1_i$ be a family of sets such that $(\bar{F}^1_i)_k$ is a set of tasks of $(\bar{\mathcal{A}}^1_i)_k$ for every k and let $\bar{F}^2_i$ be a family of sets such that $(\bar{F}^2_i)_k$ is a set of tasks of $(\bar{\mathcal{A}}^2_i)_k$ for every k, where all these tasks have infinite upper bounds. Let $(\hat{F}^1)_k$ and $(\hat{F}^2)_k$ denote $\bigcup_{i=1}^{L(k)} (\bar{F}^1_i)_k$ and $\bigcup_{i=1}^{L(k)} (\bar{F}^2_i)_k$, respectively.

Assume:

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

where p, q are polynomials and ϵ is a negligible function.

Then $(\widehat{\mathcal{A}}^1, \hat{F}^1) \leq_{\mathsf{neg,pt}} (\widehat{\mathcal{A}}^2, \hat{F}^2)$.

This sequential composition theorem can be easily extended to the case where a bounded number of components of the system are active concurrently [19].

[6] 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.

Application to 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 Sections 4 and 5. 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 (Thm. 3) and its extension to a bounded number of concurrent components, we prove that the real protocol implements the ideal protocol with respect to the long-term implementation relation $\leq_{\mathsf{neg,pt}}$. This result implies that, no matter what security failures (forgeries, guessed keys, etc.) occur up to any particular time t, new certifications and verifications performed by services that awaken after time t will still be correct (with high probability) for a polynomial-length interval of time after t.

Note that this result does *not* imply that any particular document is reliably certified for super-polynomial time. In fact, Haber's protocol does not guarantee this: even if a document certificate is refreshed frequently by new services, there is at any time a small probability that the environment guesses the current certificate, thus creating a forgery. That probability, over super-polynomial time, becomes large. Once the environment guesses a current certificate, it can continue to refresh the certificate forever, thus maintaining the forgery.

Dispatcher: We define $\mathsf{Dispatcher}_k$ for each security parameter k and set *SID*, the domain of service names, to be $\mathbb{N}$. If the environment sends a first-time certificate request, $\mathsf{Dispatcher}_k$ requests a signature from signature service j, where j is the service active at the time where this request is transmitted. After service j returns the new certificate, $\mathsf{Dispatcher}_k$ transmits it to the environment.

If a renew request for a certificate issued by the j-th signing service comes in, $\mathsf{Dispatcher}_k$ first checks to see if service j is still usable. If not, it sends a notification to the environment. Otherwise, it asks the j-th signature verification service to check the validity of the certificate. If service j answers affirmatively, $\mathsf{Dispatcher}_k$ sends a signature request to the j'-th signature service, active at the time of this request. When service j' returns, $\mathsf{Dispatcher}_k$ issues a new certificate to the environment.

Assume the following concrete time scheme. Let d be a positive natural number. Each service j is in $\mathsf{alive}(t)$ for $t = (j-1)d, \ldots, (j+2)d-1$, so j is alive in the real time interval $[(j-1)d, (j+2)d]$. Thus, at any real time t, at most three services are concurrently alive; more precisely, t lies in the interior of the intervals for at most three services. Besides, signature service j accepts signature requests for $t = (j-1)d, \ldots, jd-1$.

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 5 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 $\mathsf{Real}_k = \|_{j \in SID_k} \mathsf{RealSig}(j)_k$, $\mathsf{Ideal}_k = \|_{j \in SID_k} \mathsf{IdealSig}(j)_k$, and $\mathsf{RealSigSys}_k := \mathsf{Dispatcher}_k \| \mathsf{Real}_k$, $\mathsf{IdealSigSys}_k := \mathsf{Dispatcher}_k \| \mathsf{Ideal}_k$. Eventually, define $\overline{\mathsf{Real}} :=$

$\{\mathsf{Real}_k\}_{k\in\mathbb{N}}$, $\overline{\mathsf{Ideal}} := \{\mathsf{Ideal}_k\}_{k\in\mathbb{N}}$, $\overline{\mathsf{RealSigSys}} := \{\mathsf{RealSigSys}_k\}_{k\in\mathbb{N}}$ and $\overline{\mathsf{IdealSigSys}} := \{\mathsf{IdealSigSys}_k\}_{k\in\mathbb{N}}$. We show the following theorem.

Theorem 4. *Assume the concrete time scheme described above and assume that every signature scheme used in the timestamping protocol is complete and existentially unforgeable. Then* $(\overline{\mathsf{RealSigSys}}, \emptyset) \leq_{\mathsf{neg,pt}} (\overline{\mathsf{IdealSigSys}}, \bar{F})$*, where* $\bar{F}_k := \bigcup_{j\in SID_k}\{\{\mathsf{fail}_j\}\}$ *for every* k*.*

In order to prove this theorem, we first observe that certain components of the real and ideal systems are restricted to certain time intervals, in the sense of Definition 2: at most three $\mathsf{RealSig}(i)_k$ and $\mathsf{IdealSig}(i)_k$ services are alive at the same time. Then, we observe that the task-PIOA families $\overline{\mathsf{Real}}$ and $\overline{\mathsf{Ideal}}$ are polynomially bounded and apply the extension of our sequential composition theorem (Thm. 3) for bounded concurrency to show that $(\overline{\mathsf{Real}}, \emptyset) \leq_{\mathsf{neg,pt}} (\overline{\mathsf{Ideal}}, \bar{F})$. Eventually, using our parallel composition theorem (Thm. 2) with the $\mathsf{Dispatcher}$ automaton, we obtain the relation $(\overline{\mathsf{RealSigSys}}, \emptyset) \leq_{\mathsf{neg,pt}} (\overline{\mathsf{IdealSigSys}}, \bar{F})$, as needed.

7 Conclusion

We have introduced a new model for long-lived security protocols, based on task-PIOAs augmented with real-time task schedules. We express computational restrictions in terms of processing rates with respect to real time. The heart of our model is a long-term implementation relation, $\leq_{\mathsf{neg,pt}}$, which expresses security in any polynomial-length interval of time, despite of prior security violations. We have proved polynomial parallel composition and exponential sequential composition theorems for $\leq_{\mathsf{neg,pt}}$. Finally, we have applied the new theory to show security properties for a long-lived timestamping protocol.

This work suggests several directions for future work. First, for our particular timestamping case study, it remains to carry out the details of defining a higher-level abstract functionality specification for a long-lived timestamp service, and to use $\leq_{\mathsf{neg,pt}}$ to show that our ideal system, and hence, the real protocol, implements that specification.

We would also like to know whether or not it is possible to achieve stronger properties for long-lived timestamp services, such as reliably certifying a document for super-polynomial time.

It remains to use these definitions to study additional long-lived protocols and their security properties. The use of real time in the model should enable quantitative analysis of the rate of security degradation. Finally, it would be interesting to generalize the framework to allow the computational power of the various system components to increase with time.

References

1. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. In: Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC'85). (1985) 291–304

2. Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. In: IEEE Symposium on Security and Privacy, Oakland, CA, IEEE Computer Society (2001) 184–200
3. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In Naor, M., ed.: Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, IEEE Computer Society (2001) 136–145
4. Goldreich, O.: Foundations of Cryptography: Basic Tools. Volume 1. Cambridge University Press (2001 (reprint of 2003))
5. Canetti, R., Cheung, L., Kaynar, D., Liskov, M., Lynch, N., Pereira, O., Segala, R.: Analyzing security protocols using time-bounded Task-PIOAs. Discrete Event Dynamic Systems **18**(1) (2008) 111–159
6. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks. In: Proceedings of 10th annual ACM Symposium on Principles of Distributed Computing (PODC-91). (1991) 51–59
7. Anderson, R.: Two remarks on public key cryptology. Technical Report UCAM-CL-TR-549, University of Cambridge (2002)
8. Bellare, M., Miner, S.K.: A forward-secure digital signature scheme. In Wiener, M.J., ed.: Advances in Cryptology - CRYPTO '99. Volume 1666 of Lecture Notes in Computer Science., Springer (1999) 431–448
9. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In Biham, E., ed.: Advances in Cryptology — EUROCRYPT 2003. Number 2656 in LNCS, Springer (2003) 255–271
10. Bayer, D., Haber, S., Stornetta, S.W.: Improving the efficiency and reliability of digital time-stamping. In Capocalli, R.M., Santis, A.D., , Vaccaro, U., eds.: Sequences II: Methods in Communication, Security, and Computer Science, Springer-Verlag (1993) 329–334 (Proceedings of the Sequences Workshop, 1991).
11. Haber, S.: Long-lived digital integrity using short-lived hash functions. Technical report, HP Laboratories (2006)
12. Haber, S., Kamat, P.: 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.
13. Mitchell, J., Ramanathan, A., Scedrov, A., Teague, V.: A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. Theoretical Computer Science **353** (2006) 118–164
14. Backes, M., Pfitzmann, B., Waidner, M.: Secure asynchronous reactive systems. Cryptology ePrint Archive, Report 2004/082 (2004) `http://eprint.iacr.org/`.
15. Müller-Quade, J., Unruh, D.: Long-term security and universal composability. In: Theory of Cryptography, Proceedings of TCC 2007. Volume 4392 of LNCS., Springer-Verlag (2007) 41–60 Preprint on IACR ePrint 2006/422.
16. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing **2**(2) (1995) 250–273
17. Lynch, N., Tuttle, M.: An introduction to input/output automata. CWI Quarterly **2**(3) (1989) 219–246
18. Merritt, M., Modugno, F., Tuttle, M.: Time constrained automata. In: Proceedings of CONCUR 1991. Volume 527 of LNCS. (1991) 408–423
19. Canetti, R., Cheung, L., Kaynar, D., Lynch, N., Pereira, O.: Modeling bounded computation in long-lived systems. Cryptology ePrint Archive, Report 2007/406 (2007) `http://eprint.iacr.org/`.
20. Canetti, R., Cheung, L., Kaynar, D., Lynch, N., Pereira, O.: Compositional security for Task-PIOAs. In Sabelfeld, A., ed.: 20th IEEE Computer Security Foundations Symposium, IEEE Computer Society Press (2007) 125–139