On the Security of Long-lived Archiving Systems based on the Evidence Record Syntax

Matthias Geihs, Denise Demirel, and Johannes Buchmann

Technische Universität Darmstadt*

Abstract. The amount of security critical data that is only available in digital form is increasing constantly. The Evidence Record Syntax Specification (ERS) achieves very efficiently important security goals: integrity, authenticity, datedness, and non-repudiation. This paper supports the trustworthiness of ERS by proving ERS secure. This is done in a model presented by Canetti et al. that these authors used to establish the long-term security of the Content Integrity Service (CIS). CIS achieves the same goals as ERS but is much less efficient. We also discuss the model of Canetti et al. and propose new directions of research.

1 Introduction

The amount of data that is only available in digital form is increasing constantly. Examples include scientific data, medical records, and land registries. Therefore, digital archives are needed that efficiently and securely preserve this information for a long period of time.

Important protection goals for archived data objects are *authenticity*, *integrity*, *non-repudiation*, and *datedness*. Integrity means that the data object has not been altered. Authenticity refers to the origin being identifiable. Non-repudiation prevents an originator from repudiating that he is the origin of a document. Datedness allows to identify a time reference when a document existed.

The Evidence Record Syntax Specification (ERS) [5, 2] achieves these protection goals efficiently and in the long-term. In fact, ERS focuses on datedness. This is sufficient as integrity follows from datedness. Also, if the data objects are digitally signed, then datedness also provides authenticity and non-repudiation.

To make ERS trustworthy it is desirable to have a security model and a corresponding security proof that establishes the security properties of ERS from a theoretical point of view. This is what we do in this paper. As a security model, we use the framework of Canetti et al. for analyzing computational security in long-lived systems [4]. Using their framework, they analyze the security of the Content Integrity Service (CIS) proposed by Haber et al. [6] that also ensures datedness in archives. ERS is a refined, more efficient variant of CIS. The main

* This work has been co-funded by the DFG as part of project Long-Term Secure Archiving within the CRC 1119 CROSSING.

difference is the intelligent use of hash functions that allow for better performance. In this work, we extend their analysis of CIS to ERS. The main idea is to introduce hash services extending the signature services used by Canetti et al. They allow to model the ERS evidence records that are used to establish datedness at any point in time.

The structure of the paper is as follows. In Section 2, we describe the setup of long-term archiving systems and provide a summary of the ERS specification. In Section 3, we present the security framework of Canetti et al. and briefly explain their analysis of CIS. Using their framework, in Sections 4 and 5 we analyze the security of ERS. In Section 6, we draw conclusions and present future work.

2 ERS Archiving System

In this section, we describe the setup of secure archiving systems and provide a summary of the ERS specification.

2.1 Setup

A secure archiving system is used to store data objects for a long period of time while ensuring datedness of stored data. To achieve this, for each data object d stored at time t, the system maintains an evidence record e_d which allows to prove that data object d was archived at time t.

For maintaining evidence records, archiving systems typically rely on *timestamp services*. Timestamp services are trusted third parties which can be queried to issue a *timestamp* on a given bit string. When a timestamp service A is queried to timestamp bit string x at time t, it responds with timestamp θ. Afterwards, timestamp θ can be used to verify that timestamp service A indeed timestamped bit string x for time t.

In this work, we consider signature-based timestamp services. A timestamp for bit string x and time t issued by a signature-based timestamp service is a signature on $\langle x, t \rangle$.

2.2 ERS specification

We give an overview of the ERS specification [5]. For a set of stored data objects $\{d_1, \dots, d_n\}$, the ERS specification supports to maintain an *evidence record* e. For each data object $d \in \{d_1, \dots, d_n\}$, evidence record e can be used to verify datedness of d.

When the ERS archiving system is initially asked to store a set of data objects $\{d_1, \dots, d_n\}$, it generates a new evidence record for $\{d_1, \dots, d_n\}$ and stores it together with the data objects. The generation of an evidence record uses cryptographic primitives. In particular, collision-resistant hash functions and signature schemes are used. The lifetime of those primitives is limited due to brute-force attacks, advances in cryptanalysis, or key compromise. Consequently, in order to remain valid, an evidence record needs to be refreshed periodically.

The ERS specification provides two methods of evidence record refresh, namely timestamp-refresh and hash-refresh. Timestamp-refresh protects against the expiration of a signature-based timestamp. Hash-refresh protects against the expiration of a hash value.

We describe the data structure of an evidence record and how it is generated, timestamp-refreshed, hash-refreshed and verified.

Structure An evidence record consists of a list of timestamps and the verification information required for timestamp verification. We refrain from explicitly describing maintenance of verification information since it is not fundamental for our analysis of ERS. An initially generated evidence record contains a single timestamp. Upon evidence record refresh, new timestamps are added to the list.

Generation Generation of an evidence record e for a set of data objects $\{d_1, \ldots, d_n\}$ is done as follows. First, a Merkle hash tree [7] is generated having the data objects as the leaves. Let r be the hash value corresponding to the root of that hash tree. A timestamp θ on r is requested from a timestamp service. The freshly generated evidence record e contains timestamp θ.

Timestamp-Refresh An evidence record e is timestamp-refreshed as follows. Let $\theta_1, \ldots, \theta_n$ be the timestamps contained in e, where θ_n is the most recent timestamp. A new timestamp θ' on θ_n is requested. The timestamp-refreshed evidence record e' contains timestamps $\theta_1, \ldots, \theta_n, \theta'$.

Hash-Refresh An evidence record e is hash-refreshed as follows. Let $\{d_1, \ldots, d_n\}$ be the data objects covered by e and let $\theta_1, \ldots, \theta_n$ be the timestamps contained in e. A new Merkle hash tree is built with $d_1, \ldots, d_n, \theta_1, \ldots, \theta_n$ as the leaves. Let r' be the root of that hash tree. A new timestamp θ' on r' is requested. The hash-refreshed evidence record e' contains timestamps $\theta_1, \ldots, \theta_n, \theta'$.

Verification Datedness verification of data object d for time t_1 using evidence record e is done as follows. Let $\theta_1, \ldots, \theta_n$ be the timestamps of e and for $i = 1, \ldots, n$, let t_i be the time when θ_i was issued. Check the following.

- For $i = 2, \ldots, n$, verify if timestamp θ_i covers timestamp θ_{i-1} for time t_i. If θ_i results from hash-refresh, additionally verify if it covers data object d and timestamps $\theta_1, \ldots, \theta_{i-2}$ for time t_i.
- Verify if θ_1 covers data object d for time t_1.

3 Security Framework

In this section, we provide a high level description of the security framework of Canetti et al. for modeling computational security in long-lived systems [4]. We refer to the framework as the long-lived computational security framework, or short, LCS framework.

In this paper, our goal is to analyze the security of the ERS archiving system. In cryptography, the security of a system is typically defined in the presence of a resource bounded adversary, often modeled as a polynomial-time machine. We must allow the adversary to be active during the whole lifetime of the system. However, long-lived systems, like the ERS system, are potentially running for super-polynomial time. Modeling the adversary as a polynomial-time machine is too restrictive for analyzing the security of systems with super-polynomial lifetime.

In the context of long-lived systems, we want to allow entities to be active for unbounded lifetime, while bounding their computational power at any point in time. To model this behavior, a special kind of automaton model is used, namely the task-PIOA model [3], augmented with a notion of real time. Combining the task-PIOA model with a notion of real time allows to put in relationship the number of automaton steps and the duration of real time required to complete a task. Computational restrictions on a task-PIOA are imposed in terms of computation rates, i.e. number of computation steps per unit of real time.

By its nature, a polynomial-time machine uses only a polynomial-bounded amount of space. There is no such implicit space bound for a machine with unbounded lifetime, such as a task-PIOA. In addition to specifying a bound on the computation rate of bounded task-PIOAs, we impose a bound on the space consumed by a bounded task-PIOA. We allow a bounded task-PIOA to only use a bounded amount of space at any point in time.

Note that, with respect to the security parameter k, computational bounds are fixed over the lifetime of the whole protocol. In particular, the LCS framework does not allow to model systems whose computational power increases over time.

Using the LCS framework, a security proof of a cryptographic system is done in style of the *real-ideal paradigm*. In this style, an ideal version of the system and a real version of the system are defined. Here, the *ideal system* represents the *functionality* of the system, which is secure by definition and usually relies on a trusted party. The *real system* represents the *implementation* of the system, which uses cryptography to mimic the ideal system's behavior. To prove the implementation secure, it is shown that a computationally bounded environment interacting with the two systems cannot distinguish them. Since the ideal system implicitly defines the functionality of the secure system, this suffices to show the security of the real system.

The LCS framework provides a mechanism for long-lived systems to recover from past security failures. Therefore, an ideal system is allowed to take designated failure steps. For any polynomial-bounded time interval, the real system will only have to approximate the ideal system if no failure tasks occur in that interval.

In Section 3.1 we introduce the task-PIOA model. In Section 3.2 we introduce the long-term implementation relation which allows to compare an ideal system to a real system in the presence of a long-lived environment. In Section 3.3 we briefly describe the CIS archiving system model from [4].

3.1 Task-PIOAs

If we say, a system is described within the LCS framework, we mean that it is modeled as a *task-PIOA* [3], which is a version of a *probabilistic input/output automaton (PIOA)*.

A PIOA $\mathcal{A}$ is defined by a tuple $\langle V, S, s^{init}, I, O, H, \Delta\rangle$. Here, V is a set of state variables, S is a set of states, $s^{init} \in S$ is the initial state, I is a set of input actions, O is a set of output actions, H is a set of hidden actions, and Δ is a transition relation. The transition relation describes how the automaton, for a given action, transitions from one state into another. An action transition can be viewed as an atomic computation step of a PIOA.

Multiple PIOA actions can be grouped into a *task*. 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 locally-controlled actions (i.e., output and hidden 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 $\mathcal{A}$.

Computational bounds on a task-PIOA are three-fold. Firstly, a step bound on a task-PIOA limits the turing complexity of every single task-PIOA step. Secondly, in the LCS framework, task-PIOAs are augmented with a real-time scheduling mechanism. This allows to impose real-time scheduling constraints on task schedules. More precisely, real-time scheduling constraints allow to limit the number of steps performed by a task-PIOA per fraction of real time. Thirdly, step bound and real-time scheduling constraints are combined to obtain an overall bound.

Operations. Task-PIOAs are subject to the composition and hiding operation.

Composition. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two task-PIOAs. We say $\mathcal{A}_1$ and $\mathcal{A}_2$ are *compatible*, if they do not share any state variables or output actions, and hidden actions of the one automaton do not collide with any actions of the other automaton (and vice versa). If two task-PIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible, they can be composed into a new task-PIOA. We denote the composition of $\mathcal{A}_1$ and $\mathcal{A}_2$ by $\mathcal{A}_1 \| \mathcal{A}_2$. The composition $\mathcal{A}_1 \| \mathcal{A}_2$ is itself a task-PIOA which synchronizes on shared actions of $\mathcal{A}_1$ and $\mathcal{A}_2$.

Hiding Operator. We define a hiding operator for task-PIOAs. Let $\mathcal{A} := \langle V, S, s^{init}, I, O, H, \Delta\rangle$ be a task-PIOA and $X \subseteq O$ be a set of output actions. Then, $\mathsf{hide}(\mathcal{A}, X)$ is the task-PIOA given by $\langle V, S, s^{init}, I, O \setminus X, H \cup X, \Delta\rangle$. This prevents other task-PIOAs from synchronizing with $\mathcal{A}$ via actions in X: any task-PIOA with an action in X is no longer compatible with $\mathcal{A}$.

Step Bound. The notion of a *step bound* is defined to limit the amount of computation a task-PIOA can perform, and the amount of space it can use, in executing a single step. For $p \in \mathbb{N}$, we say a task-PIOA $\mathcal{A}$ has step bound p, if for every single step of $\mathcal{A}$, p limits the complexity of a turing machine simulating the step.

Real-time Scheduling Constraints. In the LCS framework, task-PIOAs are augmented with *real-time scheduling constraints*. This allows to model entities with unbounded lifetime but bounded processing rates. Therefore, a task schedule can be associated with a bound map $\langle rate, burst, lb, ub \rangle$. Here, $rate$ bounds the number of task executions per real time, $burst$ allows for a fixed violation of this bound, and lb and ub are lower and upper real time bounds for the first and last execution of a task, respectively. We say a real time task schedule is constrained by p, if it is valid under a p-bounded bound map.

Note that real time is only used to express constraints on task schedules. Computationally bounded system components are not allowed 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 of time in their states, based on inputs from a global task-PIOA $Clock$.

Overall Bound. Step bound and real time scheduling constraints are combined to obtain an overall bound on a task-PIOA $\mathcal{A}$. We say that a task-PIOA $\mathcal{A}$ is *p-bounded*, if $\mathcal{A}$ has step bound p and real time task scheduling is constrained by p. We say a task-PIOA $\mathcal{A}$ is *quasi-p-bounded* if $\mathcal{A}$ is of the form $\mathcal{A}' \| Clock$, where $\mathcal{A}'$ is p-bounded.

Task-PIOA Families. Task-PIOAs can be gathered into task-PIOA families, indexed by a security parameter k. A task-PIOA family $\bar{\mathcal{A}}$ is an indexed set $\{\mathcal{A}_k\}_{k \in \mathbb{N}}$ of task-PIOAs. Given a function $p : \mathbb{N} \to \mathbb{N}$, we say that $\bar{\mathcal{A}}$ is p-bounded if for all k, $\mathcal{A}_k$ is $p(k)$-bounded. If p is a polynomial, then we say $\bar{\mathcal{A}}$ is polynomially bounded.

3.2 Longterm Implementation Relation

The LCS framework allows modeling computational security in long-lived systems. Traditionally, a system is considered secure if a polynomial-time environment cannot distinguish the ideal system model (i.e., the functionality) from the real system model (i.e., the implementation). Restricting environments to be polynomial-time bounded is not satisfactory in the context of long-lived systems which potentially run for super-polynomial time.

The LCS framework provides a notion of indistinguishability in the context of long-lived systems. The idea is to not limit the overall amount of computation performed by a long-lived environment, but to polynomially bound the amount of computation performed per fraction of time. Furthermore, long-lived systems are allowed to recover from past security failures. Therefore, an ideal system is allowed to take designated failure steps. For a polynomial-bounded time interval, the real system will only have to approximate the ideal system, if no failure tasks occur in that interval.

A long-term implementation relation defines indistinguishability of systems in the context of a long-lived environment. We sketch the definition of the long-term implementation relations $\leq_{p,q,\epsilon}$ and $\leq_{neg,pt}$ given in [4], Section 5. Task-PIOAs can only be put in relationship by a long-term implementation relation if they are *comparable*. We say task-PIOAs $\mathcal{A}^1$ and $\mathcal{A}^2$ are comparable, if they have the same external interface, that is, they have the same input and output actions. We say task-PIOA families $\bar{\mathcal{A}}^1$ and $\bar{\mathcal{A}}^2$ are comparable if for every k, $(\bar{\mathcal{A}}^1)_k$ is comparable to $(\bar{\mathcal{A}}^2)_k$.

Let $\mathcal{A}^1$ and $\mathcal{A}^2$ be comparable task-PIOAs. Let F^1 and F^2 be sets of designated failure tasks associated with $\mathcal{A}^1$ and $\mathcal{A}^2$, respectively. Let $p, q \in \mathbb{N}$ and $\epsilon \in \mathbb{R}_{\geq 0}$. If for every q-bounded time window in which no failure tasks F^1 and F^2 occur, any quasi-p-bounded environment cannot distinguish $\mathcal{A}^1$ and $\mathcal{A}^2$ with probability at most ϵ, we write $(\mathcal{A}^1, F^1) \leq_{p,q,\epsilon} (\mathcal{A}^2, F^2)$.

The $\leq_{p,q,\epsilon}$ definition is extended to task-PIOA families. Let $\bar{\mathcal{A}}^1$ and $\bar{\mathcal{A}}^2$ be comparable task-PIOA families. Let $\bar{F}^1$ and $\bar{F}^2$ be sets of designated failure tasks associated with $\bar{\mathcal{A}}^1$ and $\bar{\mathcal{A}}^2$, respectively. Let p, q be polynomials and $\epsilon : \mathbb{N} \to \mathbb{R}_{\geq 0}$ be a function. We say $(\bar{\mathcal{A}}^1, \bar{F}^1) \leq_{p,q,\epsilon} (\bar{\mathcal{A}}^2, \bar{F}^2)$, if $\forall k : ((\bar{\mathcal{A}}^1)_k, (\bar{F}^1)_k) \leq_{p(k),q(k),\epsilon(k)} ((\bar{\mathcal{A}}^2)_k, (\bar{F}^2)_k)$.

We write $(\bar{\mathcal{A}}^1, \bar{F}^1) \leq_{neg,pt} (\bar{\mathcal{A}}^2, \bar{F}^2)$, 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. In this case we say $\bar{\mathcal{A}}^1$ implements $\bar{\mathcal{A}}^2$ in the sense of the long-term implementation relation. Here, $\bar{\mathcal{A}}^1$ is usually referred to as the real system (i.e., the implementation), and $\bar{\mathcal{A}}^2$ is usually referred to as the ideal system (i.e., the functionality).

Composition Theorems. We quote the following statement regarding composition theorems from [4], Section 7.

> 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 bit string 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.

Indeed, as described in Section 3.1, single task-PIOAs (e.g., encryption or timestamp services) can be composed to obtain more complex task-PIOAs (e.g., a system composed of communicating services). The following composition theorems allow to preserve the longterm implementation relation $\leq_{neg,pt}$. For a formal definition of the composition theorems see [4], Section 7.

Parallel Composition Theorem The *Parallel Composition Theorem* allows for the parallel composition of polynomially many components.

Sequential Composition Theorem The *Sequential Composition Theorem* allows for the sequential composition of exponentially many components. We say task-PIOAs are sequential if for every real time t at most one of the task-PIOAs is not dormant at time t.

d-Bounded Composition Theorem The *d-Bounded Composition Theorem* allows for the d-bounded concurrent composition of exponentially many components, where d is a positive integer. We say task-PIOAs are d-bounded concurrent if for every real time t at most d of the task-PIOAs are not dormant at time t.

We describe application of a composition theorem to sequences of task-PIOAs associated with a sequence of designated failure task families. 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 comparable sequences of compatible task-PIOA families associated with sequences of failure task set families $\bar{F}^1_1, \bar{F}^1_2, \ldots$ and $\bar{F}^2_1, \bar{F}^2_2, \ldots$, respectively. Let $C := \{1, 2, \ldots, n\}$ be a set of indices. Define the compositions of task-PIOA families $\hat{\mathcal{A}}^1 := \|_{i \in C} \bar{\mathcal{A}}^1_i$ and $\hat{\mathcal{A}}^2 := \|_{i \in C} \bar{\mathcal{A}}^2_i$, and the unions of failure task set families $\hat{F}^1 := \{\bigcup_{i \in C} (\bar{F}^1_i)_k\}_{k \in \mathbb{N}}$ and $\hat{F}^2 := \{\bigcup_{i \in C} (\bar{F}^2_i)_k\}_{k \in \mathbb{N}}$. Note that index set C is subject to the composition theorem to be applied. Then, $(\hat{\mathcal{A}}^1, \hat{F}^1) \leq_{neg,pt} (\hat{\mathcal{A}}^2, \hat{F}^2)$, if $\forall p, q \exists \epsilon \forall i : (\bar{\mathcal{A}}^1_i, \bar{F}^1_i) \leq_{neg,pt} (\bar{\mathcal{A}}^2_i, \bar{F}^2_i)$, where p, q are polynomials and ϵ is a negligible function.

3.3 CIS System Model

In [4], Canetti et al. propose a model for another long-lived archiving system, namely the content integrity service (CIS) [6]. We explain briefly how the CIS system is modeled as the composition of task-PIOAs.

The CIS system model is composed of a dispatcher component and a sequence of timestamp services. The dispatcher component accepts various timestamp requests and forwards them to the appropriate timestamp service. In [4], Section 8, it is shown that the composition of the dispatcher and real timestamp services is indistinguishable from an ideal system, composed of the same dispatcher and corresponding ideal timestamp services. 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.

We sketch some of the technicalities of the CIS analysis from [4]. The dispatcher component, the real timestamp services and the ideal timestamp services are modeled as task-PIOAs. It is shown that a real timestamp service implements its ideal timestamp service counterpart in the sense of $\leq_{neg,pt}$. Using the d-bounded composition theorem, it is shown that the d-bounded composition of real timestamp services implements the d-bounded composition of ideal timestamp services. Using the parallel composition theorem, it is shown that the parallel composition of the dispatcher and the real timestamp services (i.e., the real system) implements the parallel composition of the dispatcher and the ideal timestamp services (i.e., the ideal system).

4 ERS System Model

In this section, we propose a task-PIOA model of the ERS archiving system by extending the CIS system model (cf. Section 3.3).

The ERS system extends the CIS system as follows. The CIS system supports one method for evidence refresh, where data object and evidence are timestamped together. In particular, in the CIS system model, no hash functionality is described. The ERS system supports two methods for evidence refresh, namely timestamp-refresh and hash-refresh (cf. Section 2). The hash-refresh method is similar to CIS evidence refresh (i.e., data object and evidence are timestamped together). The timestamp-refresh method is special to ERS as it allows to refresh the evidence while only part of the current evidence needs to be hashed and timestamped. This makes ERS more efficient compared to CIS.

4.1 Construction Overview

We give an overview of the ERS model construction. The ERS system is modeled as the composition of a dispatcher component, a sequence of timestamp services, and, in particular, a sequence of hash services. The dispatcher component accepts various evidence record requests and uses appropriate hash and timestamp services to answer them.

A timestamp service can be queried to produce a timestamp for a bit string. Here, we consider signature-based timestamp services. When a signature-based timestamp service is queried for a timestamp on bit string x, it responds with a signature on $\langle x, t \rangle$, where t is the time at timestamp request. Each timestamp 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 service, which may entail a simple refresh of the timestamp key, or the adoption of a new timestamp algorithm.

A hash service can be queried to produce a hash of a bit string. When a hash service is queried for a hash of bit string x, it responds with a fixed-length hash $H(x)$, where H is a collision-resistant hash function. Because the hash service offers a collision-resistant hash functionality, it is hard to find a bit string x', such that $x \neq x'$ and $H(x) = H(x')$. Each hash service starts being available at a certain time and is available for a specified amount of time before becoming unavailable again. This can be viewed as a regular update of the hash algorithm.

The real ERS model consists of the dispatcher component, a collection of hash services, and a collection of real timestamp services. Similarly, the ideal ERS model consists of the same dispatcher component, a collection of hash services, and a collection of ideal timestamp services. Note that we do not distinguish between real and ideal hash services. This is due to the fact that we model the functionality of a collision-resistant hash algorithm using the random oracle methodology (cf. Section 4.4).

4.2 Signature Service

We describe the signature service model from [4]. A signature service is identified by its service identifier. We denote the domain of signature service identifiers by $\mathsf{SID}_{\mathsf{sign}}$. A signature service is constructed using a signature scheme.

Definition 1 (Signature scheme). *A signature scheme 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* $\mathsf{Verify}(m, \sigma, vk) = 1$*.*

In the following, we describe the real signature service model, the ideal signature service model, and sketch the proof of Theorem 1 from [4]. According to this theorem, the real signature service, if instantiated with a complete and existentially unforgeable signature scheme, implements the corresponding ideal signature service in the sense of the $\leq_{neg,pt}$ definition (cf. Section 4.2).

For every $j \in \mathsf{SID}_{\mathsf{sign}}$, suppose that $\langle \mathsf{KeyGen}_j, \mathsf{Sign}_j, \mathsf{Verify}_j\rangle$ is a signature scheme. We assume a function $\mathsf{alive} : \mathbb{T} \to 2^{\mathsf{SID}_{\mathsf{sign}}}$ 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 $\mathsf{aliveTimes}(j) := \{t \in \mathbb{T} | j \in \mathsf{alive}(t)\}$.

Real Signature Service. For $k \in \mathbb{N}$ and $j \in \mathsf{SID}_{\mathsf{sign}}$, we define three task-PIOAs, $\mathsf{KeyGen}(k, j)$, $\mathsf{Signer}(k, j)$, and $\mathsf{Verifier}(k, j)$, representing the key generator, signer, and verifier, respectively.

$\mathsf{KeyGen}(k, j)$ chooses a signing key $mySK$ and a corresponding verification key $myVK$ by running the KeyGen_j algorithm. It does this exactly once during its lifetime. It outputs the two keys 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 goes to $\mathsf{Verifier}(k, j)$. $\mathsf{Signer}(k, j)$ responds to signing requests by running the Sign_j algorithm on message m and the signing key sk. $\mathsf{Verifier}(k, j)$ accepts verification requests and simply runs the Verify_j algorithm.

For $k \in \mathbb{N}$ and $j \in \mathsf{SID}_{\mathsf{sign}}$, we define the real signature service as

$$\mathsf{RealSig}(j)_k := \mathsf{hide}(\mathsf{KeyGen}(k, j) \| \mathsf{Signer}(k, j) \| \mathsf{Verifier}(k, j), \mathsf{signKey}_j) \ .$$

Note that the hiding operator prevents the environment from learning the signing key (cf. Section 3.1).

Ideal Signature Service. We specify an ideal signature functionality $\mathsf{SigFunc}$. As with KeyGen, Signer, and $\mathsf{Verifier}$, each instance of $\mathsf{SigFunc}$ is parametrized with a security parameter k and an identifier j. The task-PIOA $\mathsf{SigFunc}(k, j)$ 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 internal variable $\mathsf{history}$, which records the set of signed messages. In addition, $\mathsf{SigFunc}(k, j)$ has an interal action fail_j, which sets a boolean flag failed. If $\mathsf{failed} = \mathsf{false}$, then $\mathsf{SigFunc}(k, j)$ uses $\mathsf{history}$ to answer verification requests: a signature is rejected if the submitted message is not in $\mathsf{history}$, even if Verify_j returns 1. If $\mathsf{failed} = \mathsf{true}$, then $\mathsf{SigFunc}(k, j)$ bypasses the check on $\mathsf{history}$, so that its answers are identical to those from the real signature service.

For $k \in \mathbb{N}$ and $j \in \mathsf{SID}_{\mathsf{sign}}$, we define the ideal signature service as

$$\mathsf{IdealSig}(j)_k := \mathsf{hide}(\mathsf{KeyGen}(k,j) \| \mathsf{SigFunc}(k,j), \mathsf{signKey}_j) \ .$$

Implementation Proof. We define standard properties of signature schemes, namely *completeness* and *existential unforgeability*. Afterwards, we show that if a real signature service is instantiated with a complete and existential unforgeable signature scheme, it implements the corresponding ideal signature service.

Definition 2 (Completeness). *A signature scheme* $\langle \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify} \rangle$ *is complete if* $\mathsf{Verify}(m, \sigma, vk) = 1$ *whenever* $\langle sk, vk \rangle \leftarrow \mathsf{KeyGen}(1^k)$ *and* $\sigma \leftarrow \mathsf{Sign}(sk, m)$.

Definition 3 (EUF-CMA). *We say a signature scheme* $\langle \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify} \rangle$ *is existentially unforgeable under adaptive chosen message attack if no polynomial-time forger has non-negligible success probability in the following game.*

Setup *The challenger runs* KeyGen *to obtain* $\langle vk, sk \rangle$ *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.*
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 $j \in \mathsf{SID}_{\mathsf{sign}}$, define the ideal signature service family

$$\overline{\mathsf{IdealSig}}(j) := \{\mathsf{IdealSig}(j)_k\}_{k \in \mathbb{N}}$$

and the real signature service family

$$\overline{\mathsf{RealSig}}(j) := \{\mathsf{RealSig}(j)_k\}_{k \in \mathbb{N}} \ .$$

Theorem 1 from [4] says that if a real signature service is instantiated with a complete and existentially unforgeable signature scheme, it implements the corresponding ideal signature service. We quote Theorem 1 from [4].

Theorem 1. *Let* $j \in \mathsf{SID}_{\mathsf{sign}}$ *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* $(\overline{\mathsf{RealSig}}(j), \emptyset) \leq_{neg,pt} (\overline{\mathsf{IdealSig}}(j), \{\mathsf{fail}_j\})$.

To prove Theorem 1, one needs to show the following for every time t and polynomials p, q. If task fail_j is not scheduled in interval $[t, t + q(k)]$, then no p-bounded environment can distinguish $\mathsf{RealSig}(j)_k$ from $\mathsf{IdealSig}(j)_k$ with high probability between time t and time $t + q(k)$. The full proof of Theorem 1 can be found in [4].

4.3 Timestamp Service

A timestamp service can be queried to create a timestamp on a bit string. The timestamp can later be used to verify that the bit string was available at a certain point in time. More precisely, for bit string x, timestamp service j can be queried to create a timestamp θ on x. The timestamp θ issued by timestamp service j is associated with a certain point in time t. Timestamp θ can later be used to verify that x was in fact timestamped for time t by service j.

We augment signature services to support timestamping. For every security parameter k and signature service $j \in \mathsf{SID}_{\mathsf{sign}}$, we define task-PIOA $\mathsf{Stamper}(k,j)$. When $\mathsf{Stamper}(k,j)$ receives a timestamp request for bit string x via action $\mathsf{reqStamp}(rid, x)$, where rid is the request identifier, it computes a signature σ on $\langle x, t\rangle$, where t is the clock reading at $\mathsf{reqStamp}$. Then, $\mathsf{Stamper}(k,j)$ responds with timestamp $\theta := \langle \sigma, t\rangle$ via $\mathsf{respStamp}(rid, \theta)$.

When $\mathsf{Stamper}(k,j)$ receives a verification request for timestamp $\theta := \langle \sigma, t\rangle$ and bit string x via $\mathsf{reqVerTs}(rid, x, \theta)$, it verifies if signature σ is a valid signature for $\langle x, t\rangle$. If verification is successful, it answers with $\mathsf{respVerTs}(rid, \mathsf{true})$. Otherwise, it answers with $\mathsf{respVerTs}(rid, \mathsf{false})$.

We use $\mathsf{Stamper}(k,j)$ and the signature service task-PIOAs defined in Section 4.2 (i.e., $\mathsf{KeyGen}(k,j)$, $\mathsf{Signer}(k,j)$, $\mathsf{Verifier}(k,j)$, and $\mathsf{SigFunc}(k,j)$) to build the real and ideal timestamp service. For $k \in \mathbb{N}$ and $j \in \mathsf{SID}_{\mathsf{sign}}$, we define the real timestamp service $\mathsf{RealStamp}(j)_k$ as

$$\begin{aligned}\mathsf{RealStamp}(j)_k := \mathsf{hide}(&\mathsf{KeyGen}(k,j) \| \mathsf{Signer}(k,j) \| \mathsf{Verifier}(k,j) \| \\ &\mathsf{Stamper}(k,j), \mathsf{signKey}_j)\end{aligned}$$

and the ideal timestamp service $\mathsf{IdealStamp}(j)_k$ as

$$\mathsf{IdealStamp}(j)_k := \mathsf{hide}(\mathsf{KeyGen}(k,j) \| \mathsf{SigFunc}(k,j) \| \mathsf{Stamper}(k,j), \mathsf{signKey}_j) \ .$$

We gather the real and ideal timestamp services into families. For $j \in \mathsf{SID}_{\mathsf{sign}}$, we define the real timestamp service family

$$\overline{\mathsf{RealStamp}}(j) := \{\mathsf{RealStamp}(j)_k\}_{k \in \mathbb{N}} \ ,$$

and the ideal timestamp service family

$$\overline{\mathsf{IdealStamp}}(j) := \{\mathsf{IdealStamp}(j)_k\}_{k \in \mathbb{N}} \ .$$

Theorem 2. *Let $j \in \mathsf{SID}_{\mathsf{sign}}$ 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 $(\overline{\mathsf{RealStamp}}(j), \emptyset) \leq_{neg,pt}$ $(\overline{\mathsf{IdealStamp}}(j), \{\mathsf{fail}_j\})$.*

Proof. By Theorem 1 we have $(\overline{\mathsf{RealSig}}(j), \emptyset) \leq_{neg,pt} (\overline{\mathsf{IdealSig}}(j), \{\mathsf{fail}_j\})$. Observe that $\overline{\mathsf{RealSig}}(j)$ and $\overline{\mathsf{IdealSig}}(j)$ are modified in the same way (i.e., pointwise composition with $\mathsf{Stamper}(k,j)$) to obtain $\overline{\mathsf{RealStamp}}(j)$ and $\overline{\mathsf{IdealStamp}}(j)$. It follows that $(\overline{\mathsf{RealStamp}}(j), \emptyset) \leq_{neg,pt} (\overline{\mathsf{IdealStamp}}(j), \{\mathsf{fail}_j\})$.

4.4 Hash Service

Generation of evidence records in the ERS system involves using hash algorithms. A hash algorithm $H : \mathcal{M} \to \mathcal{H}$ is an efficient deterministic algorithm mapping a message $m \in \mathcal{M}$ to a fixed-length hash $H(m) \in \mathcal{H}$. We call $\mathcal{M}$ the message space and $\mathcal{H}$ the hash space. We say a hash algorithm H is collision resistant if it is hard to find two messages m and m' such that $m \neq m'$ and $H(m) = H(m')$.

In order to model the functionality of a collision resistant hash algorithm we make use of the random oracle methodology [1]. A random oracle can be thought of as a public, randomly-chosen function $H : \mathcal{M} \to \mathcal{H}$ that can be evaluated only by querying an oracle that returns $H(x)$ when given input x. It can easily be seen that a random oracle serves as a collision-resistant hash algorithm. In the following, we use a random oracle in place of a collision-resistant hash functionality.

We identify a hash service by its hash service identifier. We denote the domain of hash algorithm identifiers by $\mathsf{SID}_{\mathsf{hash}}$. For security parameter $k \in \mathbb{N}$ and hash identifier $j \in \mathsf{SID}_{\mathsf{hash}}$, we define task-PIOA $\mathsf{Hasher}(k, j)$. $\mathsf{Hasher}(k, j)$ has access to a random oracle $H_{k,j} : \mathcal{M}_{k,j} \to \mathcal{H}_{k,j}$, where $|\mathcal{H}_{k,j}| \geq 2^k$. When $\mathsf{Hasher}(k, j)$ receives a hash request on message $m \in \mathcal{M}_{k,j}$ via input action $\mathsf{reqHash}(\mathit{rid}, m)$ it queries oracle $H_{k,j}$ with m and returns the hash $H_{k,j}(m) \in \mathcal{H}_{k,j}$ via output action $\mathsf{respHash}(\mathit{rid}, H_{k,j}(m))$.

In addition, $\mathsf{Hasher}(k, j)$ has an internal action fail_j, which sets a boolean flag failed. If $\mathsf{failed} = \mathsf{false}$, then $\mathsf{Hasher}(k, j)$ uses the random oracle to answer hash requests as specified above. If $\mathsf{failed} = \mathsf{true}$, then $\mathsf{Hasher}(k, j)$ denies to answer hash requests: in that case, to every request $\mathsf{reqHash}(\mathit{rid}, m)$, it responds with $\mathsf{respHash}(\mathit{rid}, \perp)$.

For $j \in \mathsf{SID}_{\mathsf{hash}}$ and security parameter k, define the hash service

$$\mathsf{Hash}(j)_k := \mathsf{Hasher}(j, k) \ .$$

For $j \in \mathsf{SID}_{\mathsf{hash}}$, define the hash service family

$$\overline{\mathsf{Hash}}(j) := \{\mathsf{Hasher}(j, k)\}_{k \in \mathbb{N}} \ .$$

4.5 Service Times

Hash services and timestamp services have limited lifetime. During protocol execution a service can be in various service states, namely being *alive*, being the *preferred* service, or being a *usable* service. Let $\mathbb{T} := \mathbb{N}$ be the domain of discrete time and define the union of all service identifiers as $\mathsf{SID} := \mathsf{SID}_{\mathsf{hash}} \cup \mathsf{SID}_{\mathsf{sign}}$. We assume the following.

– $\mathsf{alive} : \mathbb{T} \to 2^{\mathsf{SID}}$. For every t, $\mathsf{alive}(t)$ is the set of services alive at discrete time t.
– $\mathsf{aliveTimes} : \mathsf{SID} \to \mathbb{T}$. For every service j, $\mathsf{aliveTimes}(j)$ denotes the lifetime of service j, $\mathsf{aliveTimes}(j) := \{t \in \mathbb{T} : j \in \mathsf{alive}(t)\}$.

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

4.6 Dispatcher

We describe the task-PIOA $\mathsf{Dispatcher}_k$ for each security parameter k. In particular, we describe evidence record generation, timestamp-refresh, hash-refresh, and verification. In our model, an evidence record is a tuple $\langle i, \chi, \theta, j\rangle$, where i is the currently used hash service, χ is the previously timestamped data, θ is the most recent timestamp, and j is the corresponding timestamp service.

Generation. If the environment requests evidence record generation for bit string x via action $\mathsf{reqEviGen}(rid, x)$, $\mathsf{Dispatcher}_k$ requests a hash of x from hash service $i = \mathsf{pref}_{\mathsf{hash}}(t)$, where t is the clock reading at the time of the request. After hash service i returned hash h, $\mathsf{Dispatcher}_k$ requests a timestamp on $\langle i, h\rangle$ from service $j = \mathsf{pref}_{\mathsf{sign}}(t)$. After timestamp service j returned timestamp θ, $\mathsf{Dispatcher}_k$ issues a new evidence record $\langle i, x, \theta, j\rangle$ via action $\mathsf{respEvi}(rid, \langle i, x, \theta, j\rangle)$.

Timestamp-Refresh. If the environment requests timestamp-refresh of evidence record $\langle i, \chi, \theta, j\rangle$ via action $\mathsf{reqEviTs}(rid, \langle i, \chi, \theta, j\rangle)$, $\mathsf{Dispatcher}_k$ first checks to see if hash service i and timestamp service j are still usable. If not, it responds with an error message. Otherwise, it requests a hash of χ from hash service i. After hash service i returned hash h, $\mathsf{Dispatcher}_k$ checks if θ is a valid timestamp for $\langle i, h\rangle$. If not, it responds with an error message. Otherwise, it requests a hash of $\langle i, \theta\rangle$ from hash service i. After hash service i returned hash h', $\mathsf{Dispatcher}_k$ requests a timestamp on $\langle i, h'\rangle$ from service $j' = \mathsf{pref}_{\mathsf{sign}}(t)$, where t is the clock reading at the time of the request. After timestamp service j' returned timestamp θ', $\mathsf{Dispatcher}_k$ issues the refreshed evidence record $\langle i, \theta, \theta', j'\rangle$ via action $\mathsf{respEvi}(rid, \langle i, \theta, \theta', j'\rangle)$.

Hash-Refresh. If the environment requests hash-refresh of evidence record $\langle i, \chi, \theta, j\rangle$ via action $\mathsf{reqEviHash}(rid, \langle i, \chi, \theta, j\rangle)$, $\mathsf{Dispatcher}_k$ first checks to see if hash service i and timestamp service j are still usable. If not, it responds with an error message. Otherwise, it requests a hash of χ from hash service i. After hash service i returned hash h, $\mathsf{Dispatcher}_k$ checks if θ is a valid timestamp for $\langle i, h\rangle$. If not, it responds with an error message. Otherwise, it requests a hash of $\langle i, \langle x, \theta\rangle\rangle$ from hash service $i' = \mathsf{pref}_{\mathsf{hash}}(t)$, where t is the clock reading at the time of the request. After hash service i' returned hash h', $\mathsf{Dispatcher}_k$ requests a timestamp on $\langle i', h' \rangle$ from service $j' = \mathsf{pref}_{\mathsf{sign}}(t)$. After timestamp service j' returned timestamp θ', $\mathsf{Dispatcher}_k$ issues the refreshed evidence record $\langle i', \langle x, \theta \rangle, \theta', j' \rangle$ via action $\mathsf{respEvi}(rid, \langle i', \langle x, \theta \rangle, \theta', j' \rangle)$.

Verification. If the environment requests evidence verification of evidence record $\langle i, \chi, \theta, j \rangle$ via action $\mathsf{reqCheck}(rid, \langle i, \chi, \theta, j \rangle)$, $\mathsf{Dispatcher}_k$ first checks to see if hash service i and timestamp service j are still usable. If not, it responds with $\mathsf{respCheck}(rid, \mathsf{false})$. Otherwise, it requests a hash of χ from hash service i. After hash service i returned hash h, $\mathsf{Dispatcher}_k$ checks if θ is a valid timestamp for $\langle i, h \rangle$. If the verification request fails, $\mathsf{Dispatcher}_k$ responds with $\mathsf{respCheck}(rid, \mathsf{false})$. Otherwise, $\mathsf{Dispatcher}_k$ responds via action $\mathsf{respCheck}(rid, \mathsf{true})$.

4.7 ERS Service

We describe how the ideal ERS service and the real ERS service are composed of the previously described components.

Let $\mathsf{SID}_{\mathsf{hash}}$, the domain of hash service names, be $\{\mathsf{hash}\} \times \mathbb{N}$. Likewise, let $\mathsf{SID}_{\mathsf{sign}}$, the domain of timestamp service names, be $\{\mathsf{sign}\} \times \mathbb{N}$. We limit the number of service components by some exponential in security parameter k. For every k and polynomial p, let $\mathbb{N}_{<2^{p(k)}} \subseteq \mathbb{N}$ denote the set of $p(k)$-bit numbers. For every k, define service identifier subsets $(\mathsf{SID}_{\mathsf{hash}})_k \subseteq \mathsf{SID}_{\mathsf{hash}}$ and $(\mathsf{SID}_{\mathsf{sign}})_k \subseteq \mathsf{SID}_{\mathsf{sign}}$ as $(\mathsf{SID}_{\mathsf{hash}})_k := \{\mathsf{hash}\} \times \mathbb{N}_{<2^{p(k)}}$ and $(\mathsf{SID}_{\mathsf{sign}})_k := \{\mathsf{sign}\} \times \mathbb{N}_{<2^{q(k)}}$, respectively, for some polynomials p and q.

For security parameter k, define the composition of hash services

$$\mathsf{Hash}_k := \|_{j \in (\mathsf{SID}_{\mathsf{hash}})_k} \mathsf{Hasher}(k, j) \ .$$

Ideal ERS Service. The ideal ERS service is composed of a dispatcher component, a sequence of hash services, and a sequence of ideal timestamp services. For security parameter k, define the composition of ideal timestamp services $\mathsf{IdealStamp}_k := \|_{j \in (\mathsf{SID}_{\mathsf{sign}})_k} \mathsf{IdealStamp}(j)_k$. The ideal ERS service $\mathsf{IdealSys}_k$ is defined as

$$\mathsf{IdealSys}_k := \mathsf{Dispatcher}_k \| \mathsf{Hash}_k \| \mathsf{IdealStamp}_k \ .$$

Real ERS Service. The real ERS service is composed of a dispatcher component, a sequence of hash services, and a sequence of real timestamp services. For security parameter k, define the composition of real timestamp services $\mathsf{RealStamp}_k := \|_{j \in (\mathsf{SID}_{\mathsf{sign}})_k} \mathsf{RealStamp}(j)_k$. The real ERS service $\mathsf{RealSys}_k$ is defined as

$$\mathsf{RealSys}_k := \mathsf{Dispatcher}_k \| \mathsf{Hash}_k \| \mathsf{RealStamp}_k \ .$$

5 ERS Security Proof

In Section 4.7, we specified the real ERS system and the ideal ERS system. In this section, we first define a concrete time scheme according to which hash and timestamp services are active. Then, we show that the real ERS system implements the ideal ERS system in the sense of the longterm-implementation relation $\leq_{neg,pt}$.

We assume a concrete time scheme for timestamp and hash services. Let $d \in \mathbb{N}_{>0}$. Each signature service $\langle \mathsf{sign}, j \rangle \in \mathsf{SID}_{\mathsf{sign}}$ is in $\mathsf{alive}(t)$ for $t = (j-1)d, \ldots, (j+2)d - 1$, is preferred signer for times $(j-1)d, \ldots, jd-1$, and is usable for times $(j-1)d, \ldots, (j+1)d-1$. Each hash service $\langle \mathsf{hash}, j \rangle \in \mathsf{SID}_{\mathsf{hash}}$ is in $\mathsf{alive}(t)$ for $t = (j-1)de, \ldots, (j+2)de-1$, is preferred hasher for times $(j-1)de, \ldots, jde-1$, and is usable for times $(j-1)de, \ldots, (j+1)de-1$. Note that, at any real time t, at most three signature services and three hash services are concurrently alive.

Define the ideal ERS service family $\overline{\mathsf{IdealSys}} := \{\mathsf{IdealSys}_k\}_{k \in \mathbb{N}}$, and the real ERS service family $\overline{\mathsf{RealSys}} := \{\mathsf{RealSys}_k\}_{k \in \mathbb{N}}$. Let $\mathsf{SID}_k := (\mathsf{SID}_{\mathsf{hash}})_k \cup (\mathsf{SID}_{\mathsf{sign}})_k$. Define the family of empty failure sets as $\bar{\emptyset} := \{\emptyset\}_{k \in \mathbb{N}}$ and the family of signature failure sets as $\bar{F} := \{F_k\}_{k \in \mathbb{N}}$, where $F_k := \bigcup_{j \in \mathsf{SID}_k} \{\mathsf{fail}_j\}$.

Theorem 3 states that the real ERS system, $\overline{\mathsf{RealSys}}$, implements the ideal ERS system, $\overline{\mathsf{IdealSys}}$, in the sense of the long-term implementation relation $\leq_{neg,pt}$.

Theorem 3. *Assume the concrete time scheme described above and assume that every signature scheme used in the timestamping protocol is complete and existentially unforgeable. Then* $(\overline{\mathsf{RealSys}}, \bar{\emptyset}) \leq_{neg,pt} (\overline{\mathsf{IdealSys}}, \bar{F})$.

Proof. Observe that $\overline{\mathsf{RealSys}}$ and $\overline{\mathsf{IdealSys}}$ are 7-bounded concurrent and polynomially bounded. We apply the *d-Bounded Composition Theorem* to

$$\overline{\mathsf{Dispatcher}}, \overline{\mathsf{Hash}}(1), \overline{\mathsf{Hash}}(2), \ldots, \overline{\mathsf{RealStamp}}(1), \overline{\mathsf{RealStamp}}(2), \ldots$$

and

$$\overline{\mathsf{Dispatcher}}, \overline{\mathsf{Hash}}(1), \overline{\mathsf{Hash}}(2), \ldots, \overline{\mathsf{IdealStamp}}(1), \overline{\mathsf{IdealStamp}}(2), \ldots$$

to obtain $(\overline{\mathsf{RealSys}}, \bar{\emptyset}) \leq_{neg,pt} (\overline{\mathsf{IdealSys}}, \bar{F})$.

6 Conclusions

The Evidence Record Syntax specification allows to ensure datedness for data objects stored in a long-lived archiving system. We have described the Evidence Record Syntax specification and given a high level description of the LCS security framework, which is a framework for analyzing security properties of long-lived systems. Extending the CIS analysis by Canetti et al., we have analyzed the security of ERS using the LCS framework and obtained a security argument for

ERS analogous to the security argument for CIS given in [4]. This was possible because ERS is a refined, more efficient variant of CIS. In particular, we have extended the CIS analysis by introducing hash services and allowing cryptographic primitives with different lifetimes.

We now discuss in how far the security analysis of CIS and ERS establishes the expected security properties of these schemes. CIS and ERS allow for datedness verification of stored data objects. Verifiers just verify digital signatures on time stamps. They are required to trust the time stamping authorities to properly issue time stamps. They also need to trust the PKI to allow for correct signature verification.

However, the model of Canetti at al. [4] requires more trust by the retriever, namely in the archiving system to act as a trustworthy notary. This notary verifies previous time stamps and attests their validity by its signature while in the original versions of CIS and ERS all these time stamps are verified by the retrievers. Therefore, the security proof only refers to these modified versions of CIS and ERS. This is a big step forward as no security models for long-lived archiving systems were known previously. But it also raises the question of whether there is a model that allows a security proof for the original CIS and ERS. This is challenging, as the task-PIOA model only allows to process a polynomial amount of data at each point in time but over time, a super polynomial chain of time stamps may be generated.

We also discuss a few other research directions. As suggested in [4], it would be desirable to specify an abstract archiving system suiting the specification of various archiving systems such as the ERS system and the CIS system. This would allow to analyze security properties of archiving systems in a more generic way.

In this work we have been concerned with signature-based timestamping. However, other methods for timestamping exist, such as hash-linking-based timestamping. It would be worthwhile to analyze the security of such solutions.

As it has been stated in Section 8 of [4], the analysis of Canetti et al. and our results do not imply that any data object is reliably certified for super-polynomial time. This is closely related to the fact that the security parameter is fixed over the lifetime of the protocol. We would like to know if it is possible to reliably certify a document for super-polynomial time while keeping the security parameter fixed.

As it has been observed in [4] and we have stated in Section 3, the LCS framework does not allow to model components whose computational power increases over time. Since in reality, according to Moore's law and as observed over the last 40 years, computational power doubles roughly every 18 months, this seems to be a shortcoming of the framework. It might be useful to modify the framework such that it tolerates an increase of computational power over time.

Acknowledgments

This work has been co-funded by the DFG as part of project Long-Term Secure Archiving within the CRC 1119 CROSSING.

In addition, we thank Robert Künnemann for the interesting discussions.

References

[1] Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993. pp. 62–73. ACM (1993), http://doi.acm.org/10.1145/168588.168596

[2] Blazic, A.J., Saljic, S., Gondrom, T.: Extensible Markup Language Evidence Record Syntax (XMLERS). RFC 6283 (Proposed Standard) (Jul 2011), http://www.ietf.org/rfc/rfc6283.txt

[3] Canetti, R., Cheung, L., Kaynar, D.K., Liskov, M., Lynch, N.A., Pereira, O., Segala, R.: Time-bounded task-pioas: A framework for analyzing security protocols. In: Dolev, S. (ed.) Distributed Computing, 20th International Symposium, DISC 2006, Stockholm, Sweden, September 18-20, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4167, pp. 238–253. Springer (2006), http://dx.doi.org/10.1007/11864219_17

[4] Canetti, R., Cheung, L., Kaynar, D.K., Lynch, N.A., Pereira, O.: Modeling computational security in long-lived systems, version 2. IACR Cryptology ePrint Archive 2008, 492 (2008), http://eprint.iacr.org/2008/492

[5] Gondrom, T., Brandner, R., Pordesch, U.: Evidence Record Syntax (ERS) (2007), http://www.ietf.org/rfc/rfc4998.txt

[6] Haber, S.: Content Integrity Service for Long-Term Digital Archives. In: Archiving 2006. pp. 159–164. IS&T, Ottawa, Canada (2006)

[7] Merkle, R.C.: Protocols for public key cryptosystems. In: IEEE Symposium on Security and Privacy. pp. 122–134 (1980)