Long-Term Secure Time-Stamping using Preimage-Aware Hash Functions* (Full Version)

Ahto Buldas[1,2], Matthias Geihs[3], and Johannes Buchmann[3]

[1] Tallinn University of Technology, Tallinn, Estonia.
[2] Cybernetica AS, Tallinn, Estonia.
[3] Darmstadt University of Technology, Darmstadt, Germany.

Keywords: Long-Term Security, Timestamps, Preimage Aware Hash Functions

Abstract. Commonly used digital signature schemes have a limited lifetime because their security is based on computational assumptions that will potentially break in the future when more powerful computers are available. In 1993, Bayer et al. proposed a method for prolonging the lifetime of a digital signature by time-stamping the signature together with the signed document. Based on their idea long-term timestamp schemes have been developed that generate renewable timestamps. To minimize the risk of a design failure that affects the security of these schemes, it is important to formally analyze their security. However, many of the proposed schemes have not been subject to a formal security analysis yet. In this paper, we address this issue by formally analyzing the security of a hash-based long-term timestamp scheme that is based on the ideas of Bayer et al. Our analysis shows that the security level of this scheme degrades cubic over time, a security loss that needs to be taken into account when the scheme is used in practice.

1 Introduction

1.1 Motivation

More and more information is generated and stored in digital form. In many cases it is important to ensure the integrity of this information. For example, in the case of electronic health records, it is indispensable that unintentional changes to the health records can be detected. Most commonly, integrity of such sensitive information is protected using digital signature schemes. However, digital signature schemes used today can provide security only for a limited time period. Their security is based on computational assumptions, which means that they provide security only as long as the computational resources of an attacker

* This work has been co-funded by the DFG as part of project S6 within the CRC 1119 CROSSING. A short version of this paper is published in the proceedings of ProvSec 2017 [3].

are insufficient to solve a given computational problem. The widely used RSA digital signature scheme [13], for example, is broken if an attacker can find the prime factors of large integers. Using current computers, this seems to be infeasible. However, large integers can efficiently be factored using a quantum computer.

To mitigate the security risk, in 1993 Bayer et al. [1] proposed a method for prolonging the security period of a digital signature beyond the validity of the corresponding digital signature scheme. Their idea is to timestamp the signature together with the signed document in order to prove that a signature for that document was known when the corresponding signature scheme was still considered secure. Timestamps can be viewed as a digital signature themselves and have a limited security period as well. In order to protect the validity of the initial signature over a long time period, timestamps must therefore also be renewed periodically.

To understand the security of long-term integrity protection using timestamp renewal, a security model must be provided for such a scheme. Unfortunately, Bayer et al. did not provide a security model for their scheme. In fact, the security of long-term timestamp schemes has not been formally analyzed until recently. In [10], Geihs et al. analyze the security of long-term timestamp schemes that use digital signature schemes and rely on trusted timestamp services. Their analysis is done in the random oracle model and shows that the security level of such long-term timestamp schemes degrades over time. An alternative method for time-stamping uses hash functions and relies on the availability of a trusted public repository. The security of long-term timestamp schemes based on this time-stamping method has not been studied yet.

1.2 Contribution

In this work we analyze the security of long-term timestamp schemes that use hash-based timestamps. We present a security model that is based on the ideal primitive model proposed by Dodis et al. [9]. The ideal primitive model is a more refined model compared to the random oracle model used by Geihs et al. [10]. Based on the ideal primitive model, we define the notion of *extractable time-stamping*, which engages important aspects of existing security notions for timestamp schemes and at the same time appears very suitable for analyzing the security of long-term timestamp schemes. By our security analysis we establish a bound on the security level of hash-based long-term timestamp schemes that degrades cubic over time. Furthermore, we provide a framework for obtaining numeric estimations of the security loss in a practical scenario. Such a framework is valuable for engineers who design systems that rely on long-term digital evidence.

1.3 Organization

In Section 2, we briefly discuss fundamentals on cryptographic security proofs, recall the definition of preimage-aware hash functions, and finally describe time-

stamp schemes and long-term timestamp schemes in more detail. In Section 3, we define what it means for a hash-based timestamp scheme to be extractable. We show how to construct an extractable timestamp scheme using a preimage aware hash function and provide a bound on the security level of this construction. Then, in Section 4, we define what it means for a long-term timestamp scheme to be extractable and construct a hash-based long-term timestamp scheme that uses hash-based timestamp schemes for protection renewal. We prove a bound on the security level of this construction in terms of the security level of the used timestamp schemes. In Section 5, we use the results of our security analysis to evaluate the security level of hash-based long-term time-stamping in a practical scenario. In Section 6, we conclude our work and propose research directions for future work.

2 Preliminaries

2.1 Computational Security

Most commonly used cryptographic schemes provide computational security. A cryptographic primitive is *computationally secure* if for a probabilistic adversary that is given a certain amount of computational resources the probability to break the security of the scheme is negligible. Such an adversary can be thought of as a program that runs on a computing machine, formally modeled, for example, as a Turing Machine [15] or a Quantum Turing Machine [8]. The resources of such an adversary are measured in terms of the number of operation steps performed by the machine. Let p be an integer. By the class of p-step adversaries we mean all programs that halt after at most p steps.

Definition 1. *For $\epsilon : \mathbb{N} \to [0,1]$, we say a cryptographic scheme is $\epsilon(p)$-secure, if any p-step adversary breaks the scheme with probability at most $\epsilon(p)$.*

We remark that in the traditional view of cryptography the step count of an adversary is sometimes also referred to as the computation time of the adversary. In particular, no distinction between real time and computation steps is usually made. Furthermore, the class of computing machines considered is usually assumed to be the class of Turing Machines. We will see in Section 4 that for analyzing long-term security of cryptographic schemes it is useful to distinguish between real time and computational steps and to consider that the computational technology used by an adversary may change over time.

2.2 Hash Functions

A *hash function* $H\colon \{0,1\}^* \to \{0,1\}^n$ is a function that converts bitstrings of arbitrary length to n-bit digests. By a *hash chain* c, we mean a sequence $c[1], c[2], \ldots, c[m]$ of $2n$-bit strings and an m-bit string ι that is called the *shape* of c. The bits of ι are denoted by $\iota[1], \ldots, \iota[m]$. Every $c[k]$ consists of two n-bit halfs denoted by $c[k]_0$ and $c[k]_1$. By $x \overset{c}{\rightsquigarrow} r$ we mean that:

– $H(c[1]) = r$
– $H(c[k+1]) = c[k]_{\iota[k]}$, for every $k \in \{1, \ldots, m-1\}$
– $H(x) = c[m]_{\iota[m]}$

For example, a hash chain can be seen as a path through a hash tree [12] from a leaf to the root (Figure 1).

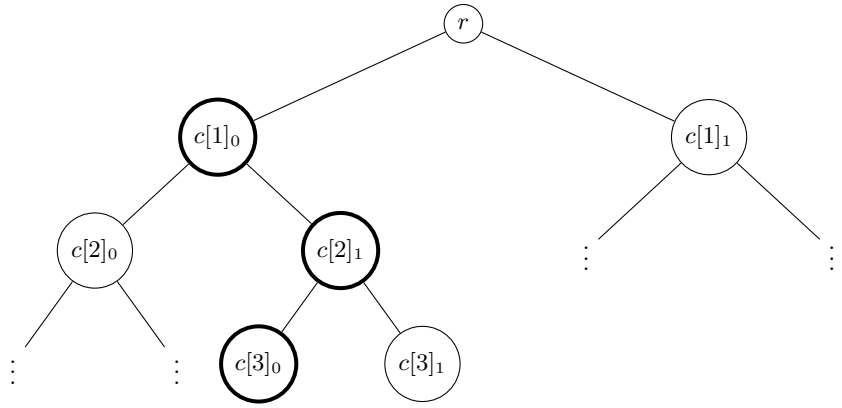

Fig. 1. A hash tree that contains hash chain $c = [c[1]_0 \| c[1]_1, c[2]_0 \| c[2]_1, c[3]_0 \| c[3]_1]$ of shape $\iota = [0, 1, 0]$, where for any x with $H(x) = c[3]_0$, $x \overset{c}{\rightsquigarrow} r$.

Preimage Awareness. Informally, a hash function H is called *preimage aware* (PrA) if whenever somebody first outputs as hash value y and later comes up with a preimage x, $H(x) = y$, then it must have known x when outputting y. The notion of preimage aware hash functions was formalized by Dodis et al. [9] for hash functions H^P that use an ideal primitive P. The primitive is ideal in the sense that it can only be called via an oracle P that records all calls made to P in an advice string adv. More formally, for H^P to be preimage aware, there must exist an efficient algorithm $\mathcal{E}$ (the so-called *extractor*) which when given y and the list adv of calls to the ideal primitive P, outputs x such that $H^P(x) = y$, or $\perp$ if the extraction failed. The adversary tries to find x and y so that $\mathcal{E}(\mathsf{adv}, y) \neq x$ and $y = H^P(x)$.

Definition 2 (Preimage Aware). *Let $\epsilon : \mathbb{N}^3 \to [0, 1]$. A function H^P is ϵ-secure* preimage aware *(*PrA*) if for every $p_{\mathcal{E}}$, p_A, and q, there is a $p_{\mathcal{E}}$-step extractor $\mathcal{E}$, such that for every p_A-step adversary A that makes at most q calls to* Ex*,*

$$\mathbf{Adv}^{\mathrm{PrA}}_{P,H}(\mathcal{A}, \mathcal{E}) = \Pr\left[\mathbf{Exp}^{\mathrm{PrA}}_{P,H}(\mathcal{A}, \mathcal{E}) = 1\right] \leq \epsilon(p_{\mathcal{E}}, p_A, q) \ .$$

Algorithm 1: The Preimage Awareness (PrA) experiment $\mathbf{Exp}^{\mathrm{PrA}}_{P,H}(\mathcal{A},\mathcal{E})$.

$x \leftarrow \mathcal{A}^{\mathsf{P},\mathsf{Ex}}$;
$y \leftarrow H^P(x)$;
if $y \in \mathtt{Q}$ **and** $\mathtt{V}[y] \neq x$ **then**
  return 1;
else
  return 0;

oracle $\mathsf{P}(m)$:
$z \leftarrow P(m)$;
$\mathsf{adv} \leftarrow \mathsf{adv}||(m,z)$;
return z;

oracle $\mathsf{Ex}(y)$:
$x \leftarrow \mathcal{E}(y, \mathsf{adv})$;
$\mathtt{Q} \leftarrow \mathtt{Q} \cup \{y\}$;
$\mathtt{V}[y] \leftarrow x$;
return x;

2.3 Time-Stamping

Digital time-stamping was first proposed by Haber and Stornetta [11]. It is used to prove that a given data object existed at a certain point in time. In the following we describe a hash-based timestamp scheme based on their ideas [11, 1] and survey various security models that have been proposed for such a scheme.

Scheme Description. The following hash-based timestamp scheme is associated with a hash function H and a set of allowed hash chain shapes $\mathcal{S}$. It uses a trusted repository Rep that accepts hash value queries. If Rep receives a hash value query r, it publishes r so that everybody can verify that r existed at this point in time t.

The time-stamping procedure is divided into rounds. During each round, a timestamp server receives a set of bitstrings $\{x_1, \ldots, x_n\}$ from clients. At the end of each round it runs algorithm Stamp to generate timestamps for these bitstrings and returns the timestamps to the clients. Algorithm Verify is used to verify timestamps.

Stamp**:** On input of bitstrings $x_1, \ldots, x_n$ ($n \leq |\mathcal{S}|$), a hash tree [12] is computed from leaves $x_1, \ldots, x_n$. Let r be the root of that hash tree and c_i be the hash chain corresponding to the path from leaf x_i to the root r (cf. Figure 1). The timestamp server publishes the root hash r at the repository Rep and for $i \in \{1, \ldots, n\}$, sends c_i as the response to request x_i. Hash chain c_i is also called a *timestamp* for bitstring x_i.

Verify**:** On input x, hash chain c, and a hash value r published at the repository, it is checked that c has allowed shape, $\mathsf{shape}(c) \in \mathcal{S}$, and c is a hash chain from x to r, $x \overset{c}{\rightsquigarrow} r$. The algorithm outputs 1 if these conditions hold, otherwise the algorithm outputs 0.

Security Model. Intuitively, security of a timestamp scheme means that an adversary cannot *back-date* any x, i.e., generate an x and a hash chain c such that $\mathsf{Verify}(x, c, r) = 1$ for an r published at Rep before the generation of x. Such a condition is formalized, for example, in [7, 5], where a two-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is considered. At the first phase of the attack, $\mathcal{A}_1$ stores hashes into the repository Rep in an arbitrary way. At the second stage, $\mathcal{A}_2$ presents

an *unpredictable* x, a hash chain c, and selects an r published at Rep so that $\mathsf{Verify}(x, c, r) = 1$. The unpredictability of x is essential as otherwise x could have been pre-computed by $\mathcal{A}_1$ before r is published and hence x could in fact be older than r.

The security definitions of [7, 5] model the future as a computationally efficient stochastic process which may not be the case in the real world. There are no arguments against the future documents having arbitrary distributions. Additionally, the success of $\mathcal{A}$ is defined as the average over such a distribution and it might still be easy to backdate fixed documents. Having such arguments in mind, extraction-based security definitions for time-stamping have been explored in [6]. Intuitively, such conditions say that whenever $\mathcal{A}_1$ publishes a hash r to Rep and later $\mathcal{A}_2$ outputs a document x and a hash chain c with $\mathsf{Verify}(x, c, r) = 1$, then x must have been "known" by $\mathcal{A}_1$ when r was stored in Rep. Formally, this is expressed by assuming the existence of an extraction algorithm $\mathcal{E}$ that depends on $\mathcal{A}_1$ and outputs a set of bitstrings X such that if $\mathcal{A}_2$ outputs (x, c) with $\mathsf{Verify}(x, c, r) = 1$ for some $r \in \mathsf{Rep}$, then $x \in X$ with overwhelming probability.

In Sections 3 and 4 we propose extraction-based security definitions for time-stamp schemes and long-term timestamp schemes in the ideal primitive model. We then analyze the security of the hash-based long-term timestamp scheme described in Section 2.4 based on these definitions.

2.4 Long-Term Time-Stamping

We describe a long-term timestamp scheme that is based on the idea of Bayer et al. to extend the lifetime of a digital signature by using timestamps [1]. Here we assume that timestamp schemes $\mathfrak{TS} = \{\mathsf{TS}_i\}_i$ are available for usage over time. Each timestamp scheme $\mathsf{TS}_i \in \mathfrak{TS}$ is associated with a start time t_i^s and a breakage time t_i^b. The start time defines when the scheme becomes available and the breakage time defines after which time the timestamps created using this scheme are not considered valid anymore. Additionally, we assume the existence of a repository Rep that is used for publishing root hash values. The long-term timestamp scheme is defined by algorithm Stamp for creating an initial timestamp, algorithm Renew for renewing a timestamp, and algorithm Verify for verifying a timestamp.

Stamp: This algorithm gets as input a timestamp scheme identifier i and a sequence of bitstrings $x_1, \ldots, x_n$. It creates timestamps for the bitstrings using scheme TS_i by computing $(r, c_1, \ldots, c_n) \leftarrow \mathsf{TS}_i.\mathsf{Stamp}(x_1, \ldots, x_n)$. Then, the root hash r is published together with identifier i at the repository Rep. Let t be the time when r was published. For $j \in \{1, \ldots, n\}$, the algorithm responds to request x_j with timestamp $T_j = [(i, c_j, r, t)]$.

Renew: This algorithm gets as input a timestamp scheme identifier i' and a sequence of bitstrings and timestamps $(x_1, T_1), \ldots, (x_n, T_n)$. The algorithm renews the timestamps using scheme $\mathsf{TS}_{i'}$ as follows. First, it computes new timestamps $(r, c_1, \ldots, c_n) \leftarrow \mathsf{TS}_{i'}.\mathsf{Stamp}(x_1 \| T_1, \ldots, x_n \| T_n)$. Then, it publishes the root hash r together with the timestamp scheme identifier i' at

the repository Rep. Let t be the time when r is published. For $j \in \{1, \ldots, n\}$, the algorithm sends (i', c_j, r, t) as the response to request (x_j, T_j). The client receiving (i', c_j, r, t) updates its timestamp by appending (i', c_j, r, t) to T_j.

Verify**:** This algorithm takes as input a bitstring x, a long-term timestamp $T = (C_1 \| \ldots \| C_n)$, where $C_j = (i_j, c_j, r_j, t_j)$, a time t, a reference $\mathtt{R}$ to the trusted repository Rep, and a set of admissible timestamp schemes $\mathfrak{TS} = \{\mathsf{TS}_i\}_i$. For $j \in \{1, \ldots, n\}$, it is verified that $\mathsf{TS}_{i_j}.\mathsf{Verify}((x \| C_1 \| \ldots \| C_{j-1}), c_j, r_j) = 1$, $(i_j, r_j) \in \mathtt{R}[t_j]$, and $t^b_{i_j} > t_{j+1}$. The algorithm outputs 1 if these conditions hold, otherwise it outputs 0.

3 Extractable Time-Stamping

In the following we define extractable time-stamping for timestamp schemes as described in Section 2.3. Informally, extractability of a timestamp scheme TS means that if a root hash r is published at the repository at time t and later someone comes up with a bitstring x and a hash chain c such that $\mathsf{TS}.\mathsf{Verify}(x, c, r) = 1$, then x must have been known at time t. Our notion of extractable time-stamping is reminiscent of PrA hash functions [9] and knowledge-binding commitments [6]. After the definition of extractable timestamp schemes, we analyze the security of the timestamp scheme from Section 2.3 instantiated using a PrA hash function.

3.1 Definition

More formally, extractability of a timestamp scheme TS^P with an ideal primitive P is defined using an experiment $\mathbf{Exp}^{\mathrm{ExTs}}$ (Algorithm 2). In this experiment an adversary $\mathcal{A}$ publishes root hashes at the repository. The adversary also uses the ideal primitive P and queries to P are recorded in an advice string adv. The definition of extractable time-stamping requires the existence of an extractor $\mathcal{E}$ with the following properties. Whenever $\mathcal{A}$ publishes a root hash r, the extractor $\mathcal{E}$ extracts from r and the advice adv, a set of supposedly timestamped bitstrings X. At the end, $\mathcal{A}$ outputs a bitstring x, a timestamp c, and a root hash r. It wins if c is valid for x and r, r was published, and x was not extracted.

Definition 3 (Extractable Time-Stamping). *Let $\epsilon : \mathbb{N}^3 \to [0, 1]$. A time-stamp scheme TS^P using ideal primitive P is ϵ-secure* extractable *(ExTs) if for all integers $p_{\mathcal{E}}$, $p_{\mathcal{A}}$, and $q_{\mathcal{E}}$, there is a $p_{\mathcal{E}}$-step extractor $\mathcal{E}$, such that for every $p_{\mathcal{A}}$-step adversary $\mathcal{A}$ that makes at most q calls to* Rep*,*

$$\mathbf{Adv}^{\mathrm{ExTs}}_{P,\mathsf{TS}}(\mathcal{A}, \mathcal{E}) = \Pr\left[\mathbf{Exp}^{\mathrm{ExTs}}_{P,\mathsf{TS}}(\mathcal{A}, \mathcal{E}) = 1\right] \leq \epsilon(p_{\mathcal{E}}, p_{\mathcal{A}}, q) \ .$$

3.2 Security Analysis

We analyze the security of the hash-based timestamp scheme from Section 2.3. We first recall various properties of hash chain shapes [4] which are useful for analyzing the security of hash-based timestamp schemes.

Algorithm 2: The extractable time-stamping experiment $\mathbf{Exp}^{\mathrm{ExTs}}_{P,\mathsf{TS}}(\mathcal{A},\mathcal{E})$.

$(x,c,r) \leftarrow \mathcal{A}^{\mathsf{P},\mathsf{Rep}}$;
if $\mathsf{TS.Verify}(x,c,r)=1$, $r \in \mathtt{R}$, **and** $x \notin \mathtt{L}[r]$ **then**
  return 1;
else
  return 0;

oracle $\mathsf{P}(m)$:
$z \leftarrow P(m)$;
$\mathsf{adv} \leftarrow \mathsf{adv}||(m,z)$;
return z;

oracle $\mathsf{Rep}(r)$:
$X \leftarrow \mathcal{E}(\mathsf{adv},r)$;
$\mathtt{R} \leftarrow \mathtt{R} \cup \{r\}$;
$\mathtt{L}[r] \leftarrow X$;
return X;

Definition 4. *We say that a timestamp scheme associated with allowed shapes $\mathcal{S}$ is N-bounded if $|\mathcal{S}| \leq N$.*

Definition 5. *An N-bounded timestamp scheme is said to be* shape-compact*, if the length of allowed hash chains does not exceed $2\log_2 N$.*

Next we proof a bound on the security of the hash-based timestamp scheme from Section 2.3 if instantiated as N-bounded and shape compact.

Theorem 1. *The timestamp scheme from Section 2.3 instantiated as N-bounded and shape compact and with an ϵ-secure* PrA *hash function H^P is ϵ'-secure extractable with*

$$\epsilon'(p_{\mathcal{E}},p_{\mathcal{A}},q) = \epsilon\left(\frac{\alpha \cdot p_{\mathcal{E}}}{2N\log_2 N}, \beta\cdot(p_{\mathcal{A}}+2qN\log_2 N), 2qN\log_2 N\right) \ ,$$

for some small constants α and β.

A detailed proof of Theorem 1 can be found in Appendix A. The $\log_2 N$ term can be eliminated if a more efficient tree-extractor is used.

Theorem 2. *The timestamp scheme from Section 2.3 instantiated as N-bounded and shape compact and with an ϵ-secure* PrA *hash function H^P is ϵ'-secure extractable with*

$$\epsilon'(p_{\mathcal{E}},p_{\mathcal{A}},q) = \epsilon\left(\frac{\alpha \cdot p_{\mathcal{E}}}{2N}, \beta\cdot(p_{\mathcal{A}}+2Nq), 2Nq\right) \ ,$$

for some small constants α and β.

Proof (Sketch). The list extractor used in the proof of Theorem 1, extracts a separate hash chain for each leaf of the hash tree. This means that each of the inner nodes of the tree are extracted many times. The efficiency of the extraction can be improved by avoiding redundant extraction of hash chains that partially overlap. This is what the tree extractor does and it leads to the improved security bound.

4 Extractable Long-Term Time-Stamping

In this section we propose a security model for the hash-based long-term timestamp scheme described in Section 2.4. First, we define a model of time and an adversary model suitable for our security analysis. Then, we define what it means for a long-term timestamp scheme to be extractable. Finally, we prove a security bound for the long-term timestamp from Section 2.4 based on the security level of the timestamp schemes used for timestamp renewal.

4.1 Model of Real Time and Computation

To model the breakage of a hash function or a hash-based timestamp scheme, we cannot use the traditional (timeless) security because it does not make sense to say that before t_i^b the hash function is secure but after t_i^b it is not secure anymore. However, in reality hash functions that have been considered secure in the past may appear insecure at a later point because the computational power of real world adversaries typically increases over time as well as new attacking algorithms, which were not known before t_i^b, may be discovered. In order to model adversaries that increase their abilities over time, we assume that the class $\mathcal{M}_t$ of computing machines available at a time t widens when t increases, i.e., $\mathcal{M}_t \subset \mathcal{M}_{t'}$ for $t < t'$. Using our approach we can model, for example, that at some point quantum computers become available.[4]

Model of Real Time. To realize our adversary model we must be able to set in relation computation time with other events (e.g., the arise of a new computational technology) happening at given points in time. Therefore, we require a model of real time in contrast to conventional security models, where time is commonly considered the same as the number of steps performed by the adversary.

In recent literature, various methods for modeling real time have been proposed. Schwenk [14] and Geihs et al. [10], for example, model real time by defining a global clock that advances whenever the adversary performs work. Another model of real time has been proposed by Canetti et al. [2]. They work in a computational framework that supports concurrency and define a global clock that runs concurrently to all other processes and ticks at a defined rate.

For our paper we follow the time formalism used by Schwenk [14] and Geihs et al. [10]. That is, we use a global clock `Clock` that holds state `time`, initialized to 0. Here, we give the adversary the control over advancing time. It may do so by calling `Clock`(t) as an oracle and if $t >$ `time`, the clock is advanced to `time` $= t$ (Algorithm 3). We remark that by advancing time, the adversary also burns computation power and triggers events in the security experiment.

[4] We remark that while our model considers quantum computing, quantum communication is not considered.

Model of Computation. As described in the previous section, we consider adversaries $\mathcal{A}^{\texttt{Clock}}$ that are associated with a global clock $\texttt{Clock}$. We bound the computational power of adversary $\mathcal{A}$ with respect to the time defined by $\texttt{Clock}$. For $\rho : \mathbb{N} \to \mathbb{N}$, we say $\mathcal{A}$ is *ρ-step-bounded* if at any time t, it performed less than $\rho(t)$ computation steps. We say $\mathcal{A}$ is *ρ-call-bounded* if at any time t, it performed less than $\rho(t)$ oracle calls.

To allow for modeling that computational technology progress over time, we define an adversary $\mathcal{A}^{\texttt{Clock}}$ as a sequence $(\mathcal{A}_0^{\texttt{Clock}}, \mathcal{A}_1^{\texttt{Clock}}, \mathcal{A}_2^{\texttt{Clock}}, \ldots)$ of machines such that $\mathcal{A}_t \in \mathcal{M}_t$, where $\mathcal{M}_t$ is the class of computing machines available at time t. Executing $\mathcal{A}^{\texttt{Clock}}$ at $\texttt{time} = t$ means executing the component $\mathcal{A}_t^{\texttt{Clock}}$. The adversary $\mathcal{A}_t^{\texttt{Clock}}$ then runs until it calls $\texttt{Clock}(t')$ after which the control is given to $\mathcal{A}_{t'}^{\texttt{Clock}}$. Here, $\mathcal{A}_{t'}^{\texttt{Clock}}$ gets access to the internal state of $\mathcal{A}_t^{\texttt{Clock}}$. An extractors $\mathcal{E}$ is also defined as a sequence $(\mathcal{E}_0, \mathcal{E}_1, \mathcal{E}_2, \ldots)$ such that $\mathcal{E}_t \in \mathcal{M}_t$, but the components do not have access to the clock-oracle. Calling an extractor $\mathcal{E}$ at $\texttt{time} = t$ means calling $\mathcal{E}_t$.

4.2 Security Definition

We define extractable long-term time-stamping using an experiment $\mathbf{Exp}^{\text{ExLTs}}$ (Algorithm 3). Similar to the definitions of PrA hash functions and extractable time-stamping without renewal, our definition of extractable long-term time-stamping uses an ideal primitive P which can only be called via an oracle P that records all calls to P in an advice string adv. The experiment also involves a global clock $\texttt{Clock}$ as described in Section 4.1.

The experiment involves an adversary $\mathcal{A}$ and an extractor $\mathcal{E}$. The adversary $\mathcal{A}$ may publish root hash values r at the repository Rep at any time by calling $\mathsf{Rep}(r)$. When Rep is called with root hash r at time t, it records r associated with t in a global table $\texttt{R}$ (i.e., $\texttt{R}[t] \leftarrow \texttt{R}[t] \cup \{r\}$, where initially $\texttt{R}[t] = \{\}$). Additionally, the extractor $\mathcal{E}$ on input adv and r extracts a set of bitstrings X that is stored associated with time t in a table $\texttt{L}$ (i.e., $\texttt{L}[t] \leftarrow \texttt{L}[t] \cup X$, where initially $\texttt{L}[t] = \{\}$). The goal of the adversary $\mathcal{A}$ is to produce (x, T, t) such that T is a valid long-term timestamp for bitstring x and time t, and x was not extracted at time t (i.e., $\mathsf{Verify}(x, c, t, \texttt{R}) = 1$ and $x \notin \texttt{L}[t]$).

Definition 6 (Extractable Long-Term Time-Stamping). *Let $\mathcal{M}$ describe the available machines classes and $\mathcal{TS}$ describe the available timestamp schemes. Let $\epsilon : \mathbb{N}^4 \to [0, 1]$. A long-term timestamp scheme LTS^P, which uses an ideal primitive P, is ϵ-secure extractable (for $\mathcal{M}$ and $\mathcal{TS}$) if for all bounds $\rho_{\mathcal{E}}$, ρ_A, and q, there is a $\rho_{\mathcal{E}}$-bounded extractor $\mathcal{E} \in \mathcal{M}$, such that for every $\rho_{\mathcal{A}}$-step-bounded and q-call-bounded adversary $\mathcal{A} \in \mathcal{M}$, and for every time t:*

$$\mathbf{Adv}^{\text{ExLTs}}_{P,\mathsf{LTS},\mathcal{TS}}(\mathcal{A}, \mathcal{E}, t) = \Pr\left[\mathbf{Exp}^{\text{ExLTs}}_{P,\mathsf{LTS},\mathcal{TS}}(\mathcal{A}, \mathcal{E}, t) = 1\right] \leq \epsilon(\rho_{\mathcal{E}}, \rho_{\mathcal{A}}, q, t) \ .$$

Algorithm 3: The extractable long-term time-stamping experiment $\mathbf{Exp}^{\text{ExLTs}}_{P,\mathsf{LTS},\mathfrak{TS}}(\mathcal{A},\mathcal{E},t^*)$.

$(x,T,t) \leftarrow \mathcal{A}^{\mathtt{Clock},\mathsf{P},\mathsf{Rep}}$;
if $\mathsf{LTS.Verify}(x,T,t,\mathtt{R},\mathfrak{TS}) = 1$, $x \notin \mathtt{L}[t]$, $\mathtt{time} \leq t^*$ **then**
  return 1;
else
  return 0;

oracle $\mathtt{Clock}(t)$:
if $t > \mathtt{time}$ **then**
  $\mathtt{time} \leftarrow t$;

oracle $\mathsf{P}(m)$:
$z \leftarrow P(m)$;
$\mathsf{adv} \leftarrow \mathsf{adv}||(m,z)$;
return z;

oracle $\mathsf{Rep}(r)$:
$X \leftarrow \mathcal{E}(\mathsf{adv},r)$;
$t \leftarrow \mathtt{time}$;
$\mathtt{R}[t] \leftarrow \mathtt{R}[t]||r$;
$\mathtt{L}[t] \leftarrow \mathtt{L}[t]||X$;
return X;

4.3 Security Analysis

Before we analyze the security of the long-term timestamp scheme described in Section 2.4, we adapt the notion of extractable time-stamping from Section 3 to the long-term setting where the class of computing machines available changes over time.

Definition 7 (Extractable Time-Stamping (Refined)). *Let $\mathcal{M}_\mathcal{E}$ and $\mathcal{M}_A$ be classes of machines and $\epsilon : \mathbb{N}^3 \to [0,1]$. We say a non-renewable timestamp scheme TS is ϵ-secure extractable for adversaries of $\mathcal{M}_\mathcal{A}$ and extractors of $\mathcal{M}_\mathcal{E}$ if for all integers $p_\mathcal{E}$, $p_\mathcal{A}$, and $q_\mathcal{E}$, there exists a $p_\mathcal{E}$-step extractor $\mathcal{E} \in \mathcal{M}_\mathcal{E}$, such that for every $p_\mathcal{A}$-step adversary $\mathcal{A} \in \mathcal{M}_\mathcal{A}$ that makes at most q calls to Rep:*

$$\mathbf{Adv}^{\text{ExTs}}_{P,\mathsf{TS}}(\mathcal{A},\mathcal{E}) \leq \epsilon(p_\mathcal{E}, p_\mathcal{A}, q) \ .$$

We now prove a bound on the security level of the long-term timestamp scheme described in Section 2.4 in terms of the security level of the available timestamp schemes.

Theorem 3. *Let $\mathcal{M}$ describe the available computing machine classes and $\mathfrak{TS} = \{\mathsf{TS}_i^P\}_i$ describe the available timestamp schemes, which use an ideal primitive P. If for every i, TS_i^P is ϵ_i-secure extractable for adversaries of $\mathcal{M}_{t_i^b}$ and extractors of $\mathcal{M}_{t_i^s}$, then the long-term timestamp scheme described in Section 2.4 is ϵ-secure extractable with*

$$\epsilon(\rho_\mathcal{E}, \rho_\mathcal{A}, q, t) = \sum_{i \in \{i : t_i^b \leq t\}} \epsilon_i\left(\alpha \cdot \rho_\mathcal{E}(t_i^b), \beta \cdot \left(\rho_\mathcal{A}(t_i^b) + q(t_i^b)\rho_\mathcal{E}(t_i^b)\right), q(t_i^b)\right) \ ,$$

for some small constants α and β.

A detailed proof of Theorem 3 can be found in Appendix B.

5 Evaluation

We evaluate which protection level the long-term timestamp scheme described in Section 2.4 provides in a practical scenario. For our evaluation we consider a scenario where data is protected over a time period of Y years. The security level of the long-term timestamp scheme is evaluated in terms of the security level of the hash functions that are used to instantiate the available timestamp schemes. Here, we assume that all used hash functions have the same security level during their validity period, where by security level we mean a bound on the success probability of an adversary.

5.1 Scenario

We assume that a set $\mathfrak{TS} = \{\mathsf{TS}_i^P\}_i$ of available hash-based timestamp schemes, where for each i, H_i^P is the hash function used by TS_i^P. We assume that the PrA-security of a hash function derives from the ratio of the adversary power p_A and the extractor power $p_{\mathcal{E}}$, and is influenced by the number of repository calls q and a base security level δ. Concretely, we assume that each hash function H_i is ϵ-secure PrA until its breakage time t_i^b with $\epsilon(p_{\mathcal{E}}, p_A, q) = \frac{p_A}{p_{\mathcal{E}}} q\delta$. Furthermore, we assume that each timestamp scheme TS_i is N-bounded and shape compact, which means that each timestamp round up to N timestamps are generated. For our practical security analysis we neglect the constants α and β derived in Theorem 2 and Theorem 3 as they are in most cases close to 1.

By Theorem 2 we obtain that each timestamp scheme TS_i is ϵ'-secure extractable until time t_i^b with

$$\epsilon'(p_{\mathcal{E}}, p_{\mathcal{A}}, q) \leq \epsilon\left(\frac{p_{\mathcal{E}}}{2N}, p_{\mathcal{A}} + 2Nq, 2Nq\right) = \frac{p_{\mathcal{A}} + 2Nq}{p_{\mathcal{E}}}(2N)^2 q\delta \ .$$

Furthermore, by Theorem 3 we obtain that the long-term timestamp scheme is ϵ''-secure long-term extractable with

$$\begin{aligned}
\epsilon''(\rho_{\mathcal{E}}, \rho_{\mathcal{A}}, q, t) &\leq \sum_{i \in I_t} \epsilon'\left(\rho_{\mathcal{E}}(t_i^b), \rho_{\mathcal{A}}(t_i^b) + q(t_i^b)\rho_{\mathcal{E}}(t_i^b), q(t_i^b)\right) \\
&\leq \sum_{i \in I_t} \frac{\rho_{\mathcal{A}}(t_i^b) + q(t_i^b)\rho_{\mathcal{E}}(t_i^b) + 2Nq(t_i^b)}{\rho_{\mathcal{E}}(t_i^b)}(2N)^2 q(t_i^b)\delta \\
&\leq \sum_{i \in I_t} \left(\frac{\rho_{\mathcal{A}}(t_i^b)}{\rho_{\mathcal{E}}(t_i^b)} + \left(\frac{2N}{\rho_{\mathcal{E}}(t_i^b)} + 1\right) q(t_i^b)\right)(2N)^2 q(t_i^b)\delta \ .
\end{aligned}$$

We assume that the adversary and the extractor have the same computation power, i.e., $\frac{\rho_{\mathcal{A}}(t)}{\rho_{\mathcal{E}}(t)} = 1$, and we observe that any reasonable extractor $\mathcal{E}$ extracts at least $2N$ bitstrings before the breakage time of a scheme, i.e., $\rho_{\mathcal{E}}(t_i^b) \geq 2N$. Let time be denoted in years and assume that each year at most L new timestamp schemes become available, and at most R root hashes are published at the repository, i.e., $|I_t| = |\{i : t_i^b \leq t\}| \leq tL$ and $q(t) \leq tR$. We obtain the following bound on the security level of the long-term timestamp scheme:

$$\epsilon''(\rho_{\mathcal{E}}, \rho_B, q, t) \leq 12t^3(NR)^2 L\delta \ .$$

5.2 Results

In Figure 2 we show the security level of the long-term timestamp scheme for different time spans Y and parameters N, L, R, and δ. The default parameters are $Y = 100$, $L = 10$, $N = 2^{32}$, $R = 365$, and $\delta = 2^{-192}$.

By the upper left graph of Figure 2, we observe a cubic security loss over time for the case that the security level of the used hash function remains constant. Using a base security level of $\delta = 2^{-192}$ for the hash function, the security level of the long-term timestamp scheme after 1 year is 2^{-104}, after 10 years it drops to 2^{-94}, and after 100 years it drops to 2^{-84}. There is a linear loss in security for increasing the number L of short-term timestamp schemes used per year, as depicted in the upper right graph. Allowing a larger number R of hash values to be published at the repository results in a quadratic security loss, as can be seen in the middle left graph. The number N of timestamps that can be issued per root hash is an important factor because in practice it may be large. The security level decreases quadratically when N is increased, as can be seen in the middle right graph. Using $N = 2^{16}$ results in security level 2^{-116} after 100 years, while using $N = 2^{48}$ results in security level 2^{-52}. Finally, the bottom graph shows how the security of the long-term timestamp scheme depends linear on the base security level δ of the used hash functions.

6 Conclusions and Future Work

We have formally analyzed the security of hash-based long-term time-stamping as proposed by Bayer et al. [1]. We prove that the security level of the discussed scheme degrades cubic over time if the security level of the used cryptographic primitives remains constant. This shows that long-lived systems need to be designed using a certain security margin. Our analysis provides a framework for analyzing the security level of similar schemes.

For future work it would be interesting to see whether the security bound proved by us can be improved and the security loss over time can be reduced. It would also be interesting to establish a security model for long-term time-stamping that does not rely on idealized assumptions such as a random oracle or an ideal primitive.

References

1. Bayer, D., Haber, S., Stornetta, W.S.: Improving the efficiency and reliability of digital time-stamping. In: Capocelli, R., De Santis, A., Vaccaro, U. (eds.) Sequences II: Methods in Communication, Security, and Computer Science. pp. 329–334. Springer New York, New York, NY (1993)
2. van Breugel, F., Chechik, M. (eds.): CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings, Lecture Notes in Computer Science, vol. 5201. Springer (2008)

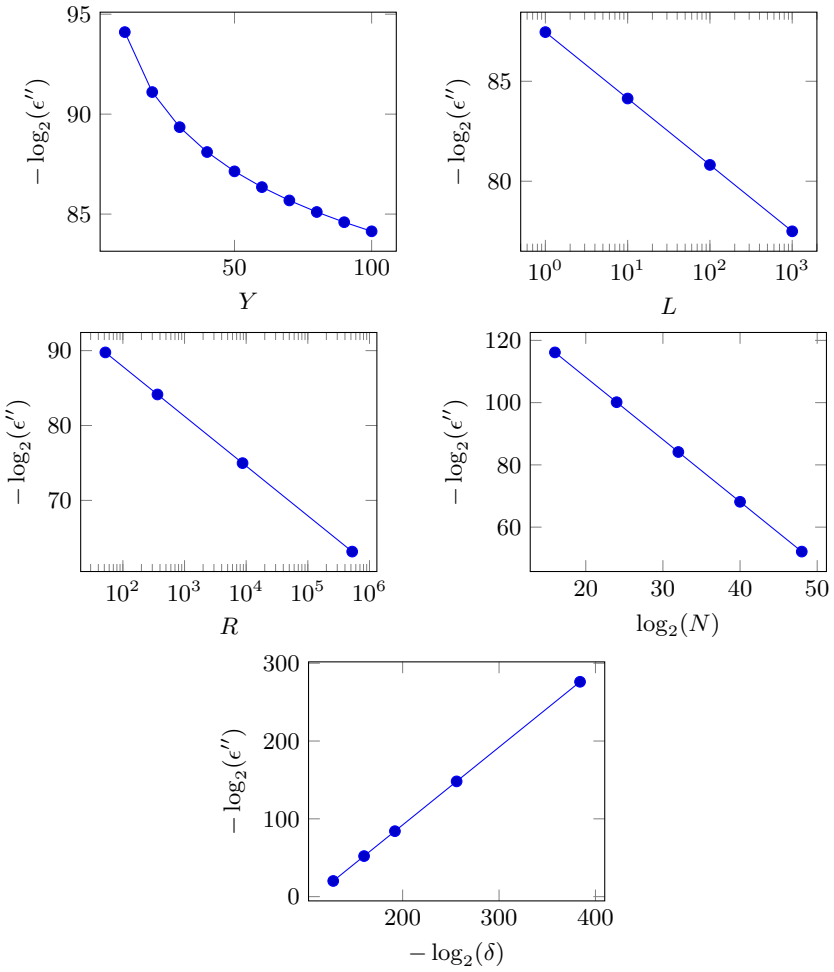

Fig. 2. Evaluation of the security level ϵ'' of long-term time-stamping when run for Y years, and with parameters L, R, N, and δ. Here, L is the number of new short-term timestamp schemes per year, R is the number of published root hashes per short-term scheme, N is the number of documents covered by one root hash, and δ is the base security level of the hash functions.

3. Buldas, A., Geihs, M., Buchmann, J.A.: Long-term secure time-stamping using preimage-aware hash functions (short version). In: Provable Security: 11th International Conference, ProvSec 2017, Xi'an, China, October 23-25, 2017, Proceedings. Springer International Publishing (2017)
4. Buldas, A., Laanoja, R.: Security proofs for hash tree time-stamping using hash functions with small output size. In: Boyd, C., Simpson, L. (eds.) Information Security and Privacy - 18th Australasian Conference, ACISP 2013, Brisbane, Australia, July 1-3, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7959, pp. 235–250. Springer (2013)
5. Buldas, A., Laur, S.: Do broken hash functions affect the security of time-stamping schemes? In: Zhou, J., Yung, M., Bao, F. (eds.) Applied Cryptography and Network Security, 4th International Conference, ACNS 2006, Singapore, June 6-9, 2006, Proceedings. Lecture Notes in Computer Science, vol. 3989, pp. 50–65 (2006)
6. Buldas, A., Laur, S.: Knowledge-binding commitments with applications in time-stamping. In: Okamoto, T., Wang, X. (eds.) Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4450, pp. 150–165. Springer (2007)
7. Buldas, A., Saarepera, M.: On provably secure time-stamping schemes. In: Lee, P.J. (ed.) Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3329, pp. 500–514. Springer (2004)
8. Deutsch, D.: Quantum theory, the church-turing principle and the universal quantum computer. Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 400(1818), 97–117 (1985)
9. Dodis, Y., Ristenpart, T., Shrimpton, T.: Salvaging merkle-damgård for practical applications. In: Joux, A. (ed.) Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5479, pp. 371–388. Springer (2009)
10. Geihs, M., Demirel, D., Buchmann, J.A.: A security analysis of techniques for long-term integrity protection. In: 14th Annual Conference on Privacy, Security and Trust, PST 2016, Auckland, New Zealand, December 12-14, 2016. pp. 449–456. IEEE (2016)
11. Haber, S., Stornetta, W.S.: How to time-stamp a digital document. In: Menezes, A., Vanstone, S.A. (eds.) Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings. Lecture Notes in Computer Science, vol. 537, pp. 437–455. Springer (1990)
12. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. Lecture Notes in Computer Science, vol. 435, pp. 218–238. Springer (1989)
13. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21(2), 120–126 (1978)
14. Schwenk, J.: Modelling time for authenticated key exchange protocols. In: Kutylowski, M., Vaidya, J. (eds.) Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II. Lecture Notes in Computer Science, vol. 8713, pp. 277–294. Springer (2014)

15. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. Proceedings of the London mathematical society 2(1), 230–265 (1937)

A Proof of Theorem 1

An N-bounded shape compact timestamp scheme using an ϵ-secure PrA hash function H^P is ϵ'-secure extractable with

$$\epsilon'(p_{\mathcal{E}}, p_{\mathcal{A}}, q) = \epsilon\left(\frac{\alpha \cdot p_{\mathcal{E}}}{2N\log_2 N}, \beta \cdot (p_{\mathcal{A}} + 2qN\log_2 N), 2qN\log_2 N\right) ,$$

for some small constants α and β.

Proof. Let $\mathcal{S}$ be the set of allowed shapes associated with an N-bounded shape compact timestamp scheme TS and let $\mathcal{E}$ be an extractor for H^P that extracts a preimage given a hash value and the ideal primitive calls. Such an extractor exists if H^P is PrA-secure. Using $\mathcal{E}$ as a black box, we construct a list extractor $\mathcal{L}$ (Algorithm 4), which extracts timestamped bitstrings from published root hash values. Having as input a P-query string adv and a bitstring r (a hash value), the list extractor $\mathcal{L}$ extracts for every allowed hash-chain shape $\iota \in \mathcal{S}$, the corresponding hash chain in a top-down way, starting from r. Due to the shape-compactness, the step count of $\mathcal{L}$ is bounded by $\alpha \cdot p_{\mathcal{E}} \cdot N \cdot \max_{\iota \in \mathcal{S}} \mathsf{length}(\iota) \leq 2 \cdot \alpha \cdot p_{\mathcal{E}} \cdot N\log_2(N)$, for some small constant α and if $p_{\mathcal{E}}$ is a bound on the step count of $\mathcal{E}$.

Algorithm 4: List extractor $\mathcal{L}(\mathsf{adv}, r)$.

```
for ι ∈ S do
    x ← E(adv, r), c ← [], k ← 0;
    while x ≠ ⊥ and k < length(ι) do
        k ← k + 1;
        x ← E(adv, x_{ι[k]}) (where x = x_0 ‖ x_1);
    L[r] ← L[r] ∪ {x};
return L;
```

Let $\mathcal{A}^{\mathsf{P},\mathsf{Rep}}$ be an adversary that participates in the ExTs-experiment denoted by $\mathbf{Exp}^{\mathrm{ExTs}}_{P,\mathsf{TS}}(\mathcal{A}, \mathcal{L})$. We construct an adversary $\mathcal{B}^{\mathsf{P},\mathsf{Ex}}$ (Algorithm 5) that participates in the PrA-experiment $\mathbf{Exp}^{\mathrm{PrA}}_{P,H}(\mathcal{B}, \mathcal{E})$ as follows. The adversary $\mathcal{B}^{\mathsf{P},\mathsf{Ex}}$ runs algorithm $\mathcal{A}^{\mathsf{P},\mathsf{Rep}}$, where calls to the repository Rep are simulated using algorithm R^{Ex} (Algorithm 6).

Algorithm 5: PrA adversary $\mathcal{B}^{\mathsf{P},\mathsf{Ex}}$.

$(x, c, r) \leftarrow \mathcal{A}^{\mathsf{P},R^{\mathsf{Ex}}}$;
$\iota \leftarrow \mathsf{shape}(c)$;
if $\exists k : \mathtt{R}[c[k]_{\iota[k]}] = 1, \mathtt{V}[c[k]_{\iota[k]}] \neq c[k+1]$ **then**
  return $c[k+1]$;
else
  return x;

Algorithm 6: Rep simulator $R^{\mathsf{Ex}}(r)$.

for $\iota \in \mathcal{S}$ **do**
  $x \leftarrow \mathsf{Ex}(r)$, $k \leftarrow 0$;
  while $x \neq \perp$ **and** $k < \mathsf{length}(\iota)$ **do**
    $k \leftarrow k + 1$;
    $x \leftarrow \mathsf{Ex}(x_{\iota[k]})$;
  $\underline{\mathtt{L}}[r] \leftarrow \underline{\mathtt{L}}[r] \cup \{x\}$;
return $\underline{\mathtt{L}}[r]$;

Note that whenever $\mathcal{A}$ succeeds in the ExTs-experiment, it finds (x, c, r) such that $x \overset{c}{\rightsquigarrow} r$, $x \in \mathtt{R}$, and $x \notin \mathtt{L}[r]$. This means there must be k such that $H^P(c[k+1]) = c[k]_{\iota[k]}$ (where ι is the shape of c), but the extractor $\mathcal{E}$ failed to extract $c[k+1]$ from $c[k]_{\iota[k]}$. Therefore, after simulating $\mathcal{A}$, the adversary $\mathcal{B}$ obtains the hash chain c and finds the smallest k, such that $\mathtt{R}[c[k]_{\iota[k]}] = 1$ and $\mathtt{V}[c[k]_{\iota[k]}] \neq c[k+1]$, and outputs $c[k+1]$, or, if no such k exists, outputs x. We observe that $\mathcal{B}$ succeeds in the PrA experiment whenever $\mathcal{A}$ succeeds in the ExTs experiment,

$$\mathbf{Adv}^{\mathrm{ExTs}}_{P,\mathsf{TS}}(\mathcal{A}, \mathcal{L}) \leq \mathbf{Adv}^{\mathrm{PrA}}_{P,H}(\mathcal{B}, \mathcal{E}) \ .$$

The step count of $\mathcal{B}^{\mathsf{P},\mathsf{Ex}}$ is $\mathcal{O}(p_{\mathcal{A}} + 2q_R N \log_2 N)$, where $p_{\mathcal{A}}$ is the step count of $\mathcal{A}$ and q_R is the number of calls to Rep. It follows that

$$\mathbf{Adv}^{\mathrm{ExTs}}_{P,\mathsf{TS}}(\mathcal{A}, \mathcal{L}) \leq \mathbf{Adv}^{\mathrm{PrA}}_{P,H}(\mathcal{B}, \mathcal{E}) \leq \epsilon(p_{\mathcal{E}}, \alpha \cdot (p_{\mathcal{A}} + 2q_R N \log_2 N), 2q_R N \log_2 N) \ ,$$

for some small constant α and where $p_{\mathcal{E}}$ is a bound on the step count of $\mathcal{E}$. The step count of $\mathcal{L}$ is $p_{\mathcal{L}} = \mathcal{O}(2p_{\mathcal{E}} N \log_2 N)$. We obtain that the timestamp scheme constructed from H^P is ϵ'-secure extractable with

$$\epsilon'(p_{\mathcal{L}}, p_{\mathcal{A}}, q_R) = \epsilon\left(\frac{\beta \cdot p_{\mathcal{L}}}{2N \log_2 N}, \alpha \cdot (p_{\mathcal{A}} + 2q_R N \log_2 N), 2q_R N \log_2 N\right) \ ,$$

for some small constant β. □

B Proof of Theorem 3

Let $\mathcal{M}$ describe the available computing machine classes and $\mathcal{TS} = \{\mathsf{TS}_i^P\}_i$ describe the available timestamp schemes, which use ideal primitive P. For every i, assume TS_i is ϵ_i-secure extractable for adversaries of $\mathcal{M}_{t_i^b}$ and extractors of $\mathcal{M}_{t_i^s}$. Then the long-term timestamp scheme described in Section 2.4 is ϵ-secure extractable with

$$\epsilon(\rho_{\mathcal{E}}, \rho_{\mathcal{A}}, q, t) = \sum_{i \in \{i : t_i^b \leq t\}} \epsilon_i\left(\alpha \cdot \rho_{\mathcal{E}}(t_i^b), \beta \cdot (\rho_{\mathcal{A}}(t_i^b) + q(t_i^b)\rho_{\mathcal{E}}(t_i^b)), q(t_i^b)\right) \ ,$$

for some small constants α and β.

Proof. Let $\mathcal{L}_i \in \mathcal{M}_{t_i^s}$ be the list extractor for TS_i that exists because TS_i is ϵ_i-secure extractable for adversaries of $\mathcal{M}_{t_i^b}$ and extractors of $\mathcal{M}_{t_i^s}$. Using the $\mathcal{L}_i$'s as black boxes, we construct a long-term list extractor $\mathcal{L} \in \mathcal{M}$ in the following way (Algorithm 7). Having as input an ideal primitive advice string adv and a bitstring r, the list extractor $\mathcal{L}$ decomposes r into a timestamp scheme identifier i and a hash value r', checks if scheme i can be used at the current time, and if so, extracts a set of bitstrings X from r' using list extractor $\mathcal{L}_i$. The step count of $\mathcal{L}$ in this run is at most the step count of $\mathcal{L}_i$ (plus a small simulation overhead).

Algorithm 7: Long-term extractor $\mathcal{L}(\mathsf{adv}, r)$

$r = (i, r')$;
if $t_i^s \leq \mathtt{time} < t_i^b$ **then**
  return $\mathcal{L}_i(\mathsf{adv}, r')$;
else
  return $\perp$;

Let $\mathcal{A}$ be any adversary that participates in the ExLTs experiment. For every i, we construct an adversary $\mathcal{B}_{(i)}$ (Algorithm 8) that participates in the ExTs experiment as follows.[5] The adversary $\mathcal{B}_{(i)}^{\mathsf{P},\mathsf{Rep}}$ simluates a run of $\mathcal{A}^{\mathtt{Clock},\mathsf{P},R}$ where the clock $\mathtt{Clock}$ is simulated using Algorithm 10 and the repository R is simulated using Algorithm 9. The simulation is performed only while $\mathtt{time}$ does not exceed the breakage time t_i^b. The adversary $\mathcal{B}_{(i)}$ tries to find a moment $t'' < t_i^b$ when $\mathcal{A}$ submits (j, r'') to the repository to renew a tuple (x', i, c', r', t') valid with scheme TS_i and $\mathcal{A}$ submitted (i, r') to the repository, but x' has not been extracted. This violates the ExLTs-condition for TS_i.

[5] We use the notation $\mathcal{B}_{(i)}$ to distinguish between $\mathcal{B}_{(i)}$ and the time-components $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1, \ldots)$ of an adversary.

Algorithm 8: ExLTs adversary $\mathcal{B}_{(i)}^{\mathsf{P},\mathsf{Rep}}$.

$(x, c, t) \leftarrow \mathcal{A}^{\mathtt{Clock},\mathsf{P},R}$ **while** $\mathtt{time} < t_i^b$
until $\exists j, x', c', r', t', r'', t''$ **such that**
- $(j, r'') \in \underline{\mathtt{R}}[t'']$ **and** $t_j^s < t'' < t_i^b, t_j^b$,
- $(x', i, c', r', t') \in \underline{\mathtt{L}}[t'']$,
- $S_i.\mathsf{Verify}(x', c', r') = 1$,
- $(i, r') \in \underline{\mathtt{R}}[t']$ **and** $t_i^s < t' < t_i^b$,
- $x' \notin \underline{\mathtt{L}}[t']$;

if $\exists$ **such** $(j, x', c', r', t', r'', t'')$ **then**
  return x';
else
  return $\perp$;

Algorithm 9: Repository simulator $R(r)$.

$r = (j, r')$;
$t \leftarrow \mathtt{time}$;
if $t_j^s \le t < t_j^b$ **then**
  if $j{=}i$ **then**
    $X \leftarrow \mathsf{Rep}(r')$;
  else
    $X \leftarrow \mathcal{L}_j(\mathsf{adv}, r')$;
else
  $X \leftarrow \perp$;
$\underline{\mathtt{R}}[t] \leftarrow \underline{\mathtt{R}}[t] \| r$;
$\underline{\mathtt{L}}[t] \leftarrow \underline{\mathtt{L}}[t] \| X$;
return X;

Algorithm 10: Clock simulator $\mathtt{Clock}(t)$.

if $t > \mathtt{time}$ **then**
  $\mathtt{time} \leftarrow t$;

Define $I_t = \{i : t_i^b \le t\}$. We observe that whenever $\mathcal{A}$ is successful until time t, one of $\{\mathcal{B}_i : t_i^b \le t\}$ is successful,

$$\mathbf{Adv}^{\mathrm{ExLTs}}_{P,\mathsf{LTS}}(\mathcal{A}, \mathcal{L}, t) \le \sum_{i \in I_t} \mathbf{Adv}^{\mathrm{ExTs}}_{P,\mathsf{TS}_i}(\mathcal{B}_{(i)}, \mathcal{L}_i) \ .$$

Assume $\mathcal{A}$ is $\rho_{\mathcal{A}}$-step-bounded and q-call-bounded, and assume $\mathcal{L}$ is $\rho_{\mathcal{L}}$-bounded. Then, for each i, the step count of $\mathcal{B}_{(i)}$ is $\mathcal{O}(\rho_{\mathcal{B}}(t_i^b) + q(t_i^b)\rho_{\mathcal{L}}(t_i^b))$. We obtain that for every t,

$$\begin{aligned}\mathbf{Adv}^{\mathrm{ExLTs}}_{P,\mathsf{LTS}}(\mathcal{A}, \mathcal{L}, t) &\le \sum_{i \in I_t} \mathbf{Adv}^{\mathrm{ExTs}}_{P,\mathsf{TS}_i}(\mathcal{B}_{(i)}, \mathcal{L}_i) \\ &\le \sum_{i \in I_t} \epsilon_i \left(\alpha \cdot \rho_{\mathcal{L}}(t_i^b), \beta \cdot (\rho_{\mathcal{B}}(t_i^b) + q(t_i^b)\rho_{\mathcal{L}}(t_i^b)), q(t_i^b)\right) \ ,\end{aligned}$$

for some small constants α and β.