Universally Composable Time-Stamping Schemes with Audit

Ahto Buldas[1,2,3,⋆], Peeter Laud[1,2,⋆⋆], Märt Saarepera, and Jan Willemson[1,4]

[1] University of Tartu, Liivi 2, 50409 Tartu, Estonia.
[2] Cybernetica, Akadeemia tee 21, 12618 Tallinn, Estonia.
[3] Tallinn University of Technology, Raja 15, 12618 Tallinn, Estonia.
[4] Playtech Estonia.

Abstract. We present a universally composable time-stamping scheme based on universal one-way hash functions. The model we use contains an ideal auditing functionality (implementable in the Common Reference String model), the task of which is to check that the rounds' digests are correctly computed. Our scheme uses hash-trees and is just a slight modification of the known schemes of Haber-Stornetta and Benaloh-de Mare, but both the modifications and the audit functionality are crucial for provable security. The scheme turns out to be nearly optimal – we prove that in every universally composable auditable time-stamping scheme, almost all time stamp requests must be communicated to the auditor.

Keywords: Time-Stamping, Provable Security, One-Way Hash Functions, Universal Composability.

1 Introduction

Time-stamping is an important data integrity protection mechanism the main objective of which is to prove that electronic records existed at certain time. The scope of applications of time-stamping is very large and the combined risks related to time stamps are potentially unbounded. Hence, the standard of security for time-stamping schemes must be very high. It is highly unlikely that currently popular trusted third party solutions are sufficient for all needs, since the practice has shown that insider threats by far exceed the outside ones. This motivates the development of time-stamping schemes that are provably secure even against malicious insiders.

Several constructions of potentially insider-resistant time-stamping schemes have been proposed [6, 14, 15, 7, 19] based on collision-resistant hash functions. However, only few analytical arguments confirm the security of these schemes. Two early attempts to sketch a security proof [6, 15] were recently shown to be flawed [8]. Presently, there are two schemes with correct security proofs: a non-interactive time-stamping scheme in the bounded storage model [19] and a bounded hash-chain scheme in the standard model [8]. However, the schemes in use (like [25–27]) still have no formal security proofs.

The formal security conditions for time-stamping schemes are still a subject under discussion. The early works [6, 14, 15] focused on the *consistency of databases* maintained by time-stamping service providers. It was required to be hard to change the database without compromising its consistency with a digest published in a secure repository. In [8] it was pointed out that one of the implicit assumptions of the consistency condition – the adversary knows at least one pre-image of a published digest – may be unjustified for malicious service providers. An independent security condition was proposed [8] in which the stream of time stamp requests is modeled as a high-entropy distribution. Considering the wide range of time-stamping applications, it cannot be taken for granted that these two conditions are sufficient. *Universal Composability (UC) framework* [1–4, 9, 21–23] provides a more general approach to security – rather than studying *ad hoc* behavior of adversaries, it is proved that real security primitives faithfully implement a certain *ideal primitive* the security of which is evident. Therefore, *all* security features of the ideal primitive (including the *ad hoc* ones mentioned above) are transferred to the real primitive.

In this paper, we construct universally composable time-stamping schemes under an assumption that they contain a third party auditing functionality. The idea of third-party audit in time-stamping schemes is

⋆ Supported by Estonian SF grant no. 5870
⋆⋆ Supported by Estonian SF grant no. 6095

natural and certainly not new. It has been proposed as one of the additional security measures in commercial time-stamping schemes [25]. Still, the formal security conditions presented thus far do not include the audit explicitly. We include audit functions into a general time-stamping scheme and present new security conditions that reflect two different types of auditability – audit-supported publishing and multi-round audit. We present a practical construction of an auditable time-stamping scheme that uses slightly modified Merkle trees [18] and collision resistant (or universal one-way) hash functions. We prove that the scheme is secure in the sense of conventional security conditions, assuming that the underlaying hash function is collision-resistant. The auditor is crucial in the scheme – the negative results in [8] imply that the ordinary reduction techniques are insufficient for such proofs in case no additional functionalities are added to the time-stamping scheme.

We also prove that our construction of a time-stamping scheme with audit-supported publishing is universally composable, if the hash function used is universally one-way. Our construction turns out to be nearly optimal in the sense of communication between the time-stamping service and the auditor.

In Section 2, we present notations and definitions. In Section 3, we define auditable time-stamping schemes and the corresponding security notions. In Section 4, we construct an auditable time-stamping scheme based on collision-resistant hash functions and prove that the construction is secure. In Section 5, we prove that our construction gives a universally composable time-stamping scheme with audit-supported publishing. In Section 6, we prove that our construction is nearly optimal.

2 Notation and Definitions

By $x \leftarrow \mathcal{D}$ we mean that x is chosen randomly according to a distribution $\mathcal{D}$. If A is a probabilistic function or a Turing machine, then $x \leftarrow \mathsf{A}(y)$ means that x is chosen according to the output distribution of A on an input y. By $\mathcal{U}_n$ we denote the uniform distribution on $\{0,1\}^n$. If $\mathcal{D}_1, \ldots, \mathcal{D}_m$ are distributions and $F(x_1, \ldots, x_m)$ is a predicate, then $\mathsf{Pr}[x_1 \leftarrow \mathcal{D}_1, \ldots, x_m \leftarrow \mathcal{D}_m \colon F(x_1, \ldots, x_m)]$ denotes the probability that $F(x_1, \ldots, x_m)$ is true after the ordered assignment of $x_1, \ldots, x_m$. For functions $f, g \colon \mathbb{N} \to \mathbb{R}$, we write $f(k) = O(g(k))$ if there are $c, k_0 \in \mathbb{R}$, so that $f(k) \leq cg(k)$ ($\forall k > k_0$). We write $f(k) = \omega(g(k))$ if $\lim\limits_{k\to\infty} \frac{g(k)}{f(k)} = 0$. If $f(k) = k^{-\omega(1)}$, then f is *negligible*. A Turing machine M is *polynomial-time* (*poly-time*) if it runs in time $k^{O(1)}$, where k denotes the input size. Let FP be the class of all functions $f \colon \{0,1\}^* \to \{0,1\}^*$ computable by a poly-time M. A distribution $\mathcal{D}$ on $\{0,1\}^*$ is *polynomially sampleable* if it is an output distribution of a poly-time Turing machine. A polynomially sampleable distribution $\mathcal{D}$ is *polynomially unpredictable* if $\mathsf{Pr}[L \leftarrow \Pi(1^k), x \leftarrow \mathcal{D} \colon x \in L] = k^{-\omega(1)}$ for every predictor $\Pi \in \mathsf{FP}$. We say that $\mathcal{D}_1$ and $\mathcal{D}_2$ (on $\{0,1\}^*$) are *indistinguishable* (and write $\mathcal{D}_1 \approx \mathcal{D}_2$) if $|\ \mathsf{Pr}[x \leftarrow \mathcal{D}_1 \colon \Delta(1^k, x) = 1] - \mathsf{Pr}[x \leftarrow \mathcal{D}_2 \colon \Delta(1^k, x) = 1]\ | = k^{-\omega(1)}$ for every distinguisher $\Delta \in \mathsf{FP}$.

A *collision-resistant hash function* is a family $\{h_k \colon \{0,1\}^* \to \{0,1\}^k\}_{k\in\mathbb{N}}$, such that $\delta(k) = \mathsf{Pr}[h_k \leftarrow \mathfrak{F}, (x, x') \leftarrow \mathsf{A}(1^k, h_k) \colon x \neq x',\ h_k(x) = h_k(x')] = k^{-\omega(1)}$ for every $\mathsf{A} \in \mathsf{FP}$. Here, $\mathfrak{F}$ is a certain poly-sampleable distribution on F^*. We write $h(x)$ instead of $h_k(x)$.

A *Universal One-Way Hash Function* (*UOWHF*) is a family $\{h_k \colon \{0,1\}^* \to \{0,1\}^k\}_{k\in\mathbb{N}}$, such that $\mathsf{Pr}[x \leftarrow \mathsf{A}_1(1^k), h_k \leftarrow \mathfrak{F}, x' \leftarrow \mathsf{A}_2(h_k, x) \colon x \neq x',\ h_k(x) = h_k(x')] = k^{-\omega(1)}$ for every $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2) \in \mathsf{FP}$. It is proven [20] that UOWHFs can be constructed from one-way functions.

3 Auditable Time-Stamping Schemes

The term *time-stamping* is somewhat misleading. In academic papers [5–8, 14, 15], time-stamping schemes are treated as secure logging mechanisms, rather than protocols for assigning precise time values to data items. The main motivation of the time-stamping schemes discussed in this paper is related to integrity protection (and preservation). For example, time stamps can prove that a digital signature was created before the corresponding certificate was revoked. For applications like this, the precision of one day is mostly sufficient. Applications that require high-precision time are out of the scope of this paper.

3.1 General Definition of a Time-Stamping Scheme

A time-stamping scheme TS is capable of: (1) assigning a time-value $t \in \mathbb{N}$ to each request $x \in \{0,1\}^k$, and (2) verifying whether x was time-stamped during the t-th time unit (hour, day, week, etc.). Almost all known time-stamping schemes consist of the following component-processes:

– Repository – a write only database that receives k-bit digests, adds them to a list $\mathfrak{D}$. Repository also receives queries $\tau \in \mathbb{N}$ and returns $\mathfrak{D}[\tau]$ if $\tau \leq |\mathfrak{D}|$. Otherwise, Repository returns NIL. We assume that the repository is updated in a regular way (say daily), and the update time/date is known to the users of the system. This is a link between the real time and the modeled time value $t = |\mathfrak{D}|$. Practical schemes [25] use newspaper-publishing as the Repository. Therefore, it is reasonable to assume that Repository is costly and to keep the number of stored bits as small as possible.
– Stamper – operates in discrete time intervals called *rounds*. During a t-th round, Stamper receives requests x and returns pairs (x,t). We assume that Stamper "knows" how many digests have been stored to Repository. Let L_t be the list of all requests received during the t-th round. In the end of the round, Stamper creates a *certificate* $c = \mathsf{Stamp}(x; L_t, L_{t-1}, \ldots, L_1)$ for each request $x \in L_t$. Besides, Stamper computes a digest $d_t = \mathsf{Publish}(L_t, \ldots, L_1)$ and sends d_t to Repository.
– Verifier – a computing environment for verifying time stamps. In practice, each user may have its own Verifier but for the security analysis, it is sufficient to have only one. It is assumed that Verifier has a tamperproof access to Repository. On input (x,t), Verifier obtains a certificate c from Stamper, and a digest $d = \mathfrak{D}[t]$ from Repository, and returns $\mathsf{Verify}(x,c,d) \in \{\mathsf{yes}, \mathsf{no}\}$. It is not specified how c is transmitted from Stamper to Verifier. In practice, c can be stored together with x. Hence, the size of c should be reasonable. Note that x can be verified only after the digest d_t is sent to Repository. This is acceptable, because in the applications we address, x is verified long after stamping.
– Client – any application-environment that uses Stamper and Verifier.

Definition 1 (Correctness). *A triple of functions* $\mathsf{TS} = (\mathsf{Stamp}, \mathsf{Publish}, \mathsf{Verify})$ *is called a* time-stamping scheme *if* $\mathsf{Verify}(x_n, \mathsf{Stamp}(x, \mathcal{L}), \mathsf{Publish}(\mathcal{L})) = \mathsf{yes}$ *for every* $\mathcal{L} = (L_t, \ldots, L_1)$*, and* $x \in L_t$.

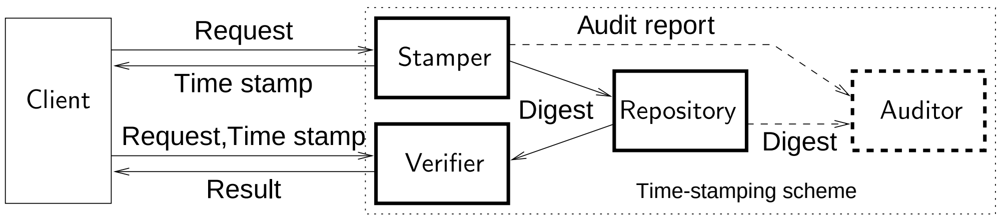

Fig. 1. General view of a time-stamping scheme.

3.2 Security Conditions

It is assumed that an adversary A is able to corrupt Stamper, some instances of Client and some instances of Verifier. The Repository is assumed to be non-corrupting. After closing the t-th round (i.e. after publishing d_t) it should be impossible to add a new request x to the set L_t of requests and prove to a Verifier that $x \in L_t$ by finding a suitable certificate c. This suggests the following security condition:

Definition 2 (Consistency). *A time-stamping scheme is* consistent *if for every* $\mathsf{A} \in \mathsf{FP}$*:*

$$\Pr[(L_t, \ldots, L_1, c, x) \leftarrow \mathsf{A}(1^k) \colon x \notin L_t,\ \mathsf{Verify}(x, c, \mathsf{Publish}(L_t, \ldots, L_1)) = \mathsf{yes}] = k^{-\omega(1)} \ . \quad (1)$$

The condition (1) is not completely satisfactory because the adversary has to explicitly construct the lists $L_t, \ldots, L_1$ of time-stamped requests. Back-dating attacks can be possible without A creating these lists. For example, A may publish a value d which is not necessarily computed by using the $\mathsf{Publish}$ function and then, after obtaining a new (randomly generated) x, to find a certificate so that $\mathsf{Verify}(x, c, d) = \mathsf{yes}$. This suggests a somewhat different security condition [8]:

Definition 3 (Security against random back-dating). *A time-stamping scheme is* secure against random back-dating *if for every polynomially unpredictable distribution $\mathcal{D}$ on $\{0,1\}^k$ and $(\mathsf{A}_1, \mathsf{A}_2) \in \mathsf{FP}$:*

$$\Pr[(d,a)\leftarrow\mathsf{A}_1(1^k), x\leftarrow\mathcal{D}, c\leftarrow\mathsf{A}_2(x,a) \colon \mathsf{Verify}(x,c,d)=\mathsf{yes}] = k^{-\omega(1)} \ . \tag{2}$$

In some applications, additional security features (like confidentiality of messages, availability etc.) of time-stamping schemes are needed. This paper does not study these features. To show why the security of time-stamping is an issue of independent importance, we point out the essential differences between time-stamping and *commitment schemes*.

Time-Stamping and Commitment Schemes. The security requirements of *Commitment Schemes* differ from those of time-stamping in various aspects. In principle, time-stamping schemes can be implemented by using commitment schemes, but as the requirements to time-stamping schemes are much lower, this would be inefficient for real applications. The main differences between these schemes are the following:

– **Message Secrecy**. One of the two main requirements to Commitment Schemes is *message secrecy* – the Receiver should not be able to read the committed message before the message is opened by the Committer. In Time-Stamping Schemes, message secrecy is not the primary security property, though in some applications (like patent filing) it would be desirable.
– **Commitment Size**. In time-stamping schemes, the commitment size is independent of the message size and is much shorter than the message. This is mostly not the case for Commitment Schemes.
– **Selective Release**. In time-stamping schemes, a committed message m is a list of submessages $m_1, \ldots, m_n$. During the verification, only one submessage m_i is ”opened” by presenting the corresponding certificate c_i to the verifier. In Commitment Schemes, the whole message m is opened.

3.3 Time-Stamping Schemes with Audit

It is essential for the security of time-stamping that a corrupted $\mathsf{Stamper}$ is not able to publish a value d in $\mathsf{Repository}$ without actually knowing a database $(x_1, \ldots, x_n)$ such that $\mathsf{Publish}(x_1, \ldots, x_n) = d$. Otherwise, it could be difficult (or even impossible) to find a security proof [8]. The easiest way of proving such knowledge is sending the requests $x_1, \ldots, x_n$ to a trusted $\mathsf{Auditor}$ who checks whether $\mathsf{Publish}(x_1, \ldots, x_n) = d$. Such an audit procedure may be performed before publishing or after publishing. We observe two different audit models:

– *Audit-Supported Publishing*. In this model, the roles of $\mathsf{Repository}$ and $\mathsf{Auditor}$ are merged. If the t-th round is closed, the $\mathsf{Auditor/Repository}$ receives a list L_t of bit-strings and an audit report from $\mathsf{Stamper}$ and checks their correctness. The digest is not published if the audit report is incorrect.
– *Multi-Round Audit*. In this model, audit reports are checked long after publishing, which is much more close to the real-life audit, which is performed no more than once a year.

We define two additional (audit) functions: Report for creating a report $r_t = \mathsf{Report}(L_t, \ldots, L_1)$, and Audit for verifying a report by checking the consistency of r_t and $d_t = \mathsf{Publish}(L_t, \ldots, L_1)$.

Definition 4. *A 5-tuple* $\mathsf{ATS} = (\mathsf{Stamp}, \mathsf{Publish}, \mathsf{Verify}, \mathsf{Report}, \mathsf{Audit})$ *is called an* auditable time-stamping scheme *if* $\mathsf{Audit}(\mathsf{Report}(\mathcal{L}), \mathsf{Publish}(\mathcal{L})) = \mathsf{yes}$, *for any $\mathcal{L} = (L_t, \ldots, L_1)$ (properly created audit reports verify successfully), and* $(\mathsf{Stamp}, \mathsf{Publish}, \mathsf{Verify})$ *is a time-stamping scheme.*

In the schemes analyzed in this paper, we assume that the value of $\mathsf{Audit}(L_t, \ldots, L_1)$ depends only on the first argument L_t. The results we obtain for such schemes can be easily generalized.

Schemes with Audit-Supported Publishing. The audit is performed during (or before) publishing. The auditor is a trusted middle-man between $\mathsf{Stamper}$ and $\mathsf{Publisher}$. After the t-th round, $\mathsf{Stamper}$ computes a digest $d_t = \mathsf{Publish}(L_t, \dots, L_1)$ and an audit report $r_t = \mathsf{Report}(L_t, \dots, L_1)$. Having sent a pair (d, r), the auditor checks whether $\mathsf{Audit}(r, d) = \mathsf{yes}$ and sends d to $\mathsf{Repository}$. Hence, a successful publishing is possible only if a correct audit report is sent to the auditor. A time-stamping scheme with audit-supported publishing is secure against random back-dating if for every polynomially unpredictable distribution $\mathcal{D}$ and for every $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2) \in \mathsf{FP}$:

$$\mathsf{Pr}[(d, r, a) \leftarrow \mathsf{A}_1(1^k), x \leftarrow \mathcal{D}, c \leftarrow \mathsf{A}_2(x, a) \colon \mathsf{Verify}(x, c, d) = \mathsf{yes} = \mathsf{Audit}(r, d)] = k^{-\omega(1)} \ . \tag{3}$$

Schemes with Multi-Round Audit. Publishing is done like in the schemes without audit. The audit function is performed after publishing. If N rounds are passed, $\mathsf{Stamper}$ computes audit reports $r_1 = \mathsf{Report}(L_1)$, ..., $r_N = \mathsf{Report}(L_N, \dots, L_1)$ and sends $(r_1, \dots, r_N)$ to the auditor. For every $t = 1 \dots N$, the auditor obtains d_t from $\mathsf{Repository}$ and computes $\mathsf{Audit}(r_t, d_t)$. If for some t the result is no then all users are informed about the incident. A time-stamping scheme with multi-round audit is secure against random back-dating if for every polynomially unpredictable $\mathcal{D}$ and for every $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2) \in \mathsf{FP}$:

$$\mathsf{Pr}[(d, a) \leftarrow \mathsf{A}_1(1^k), x \leftarrow \mathcal{D}, (c, r) \leftarrow \mathsf{A}_2(x, a) \colon \mathsf{Verify}(x, c, d) = \mathsf{yes} = \mathsf{Audit}(r, d)] = k^{-\omega(1)} \ . \tag{4}$$

3.4 Records of Arbitrary Length

The definitions above assume that all time stamp requests are k bits long. To time-stamp longer records, practical schemes use collision-resistant hash functions (at the client side) to make requests shorter. Since these hash functions have influence on the security, they have to show up in the security conditions.

Definition 5. *A time-stamping scheme with audit-supported publishing is said to be secure relative to a client side hash function $h \colon \{0,1\}^* \to \{0,1\}^k$ if for every polynomially unpredictable distribution $\mathcal{D}$ on $\{0,1\}^*$ and for every $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2) \in \mathsf{FP}$:*

$$\mathsf{Pr}[(d, r, a) \leftarrow \mathsf{A}_1(1^k), X \leftarrow \mathcal{D}, c \leftarrow \mathsf{A}_2(X, a) \colon \mathsf{Verify}(h(X), c, d) = \mathsf{yes} = \mathsf{Audit}(r, d)] = k^{-\omega(1)} \ . \tag{5}$$

A time-stamping scheme with audit-supported publishing is said to be secure relative to a client side hash function h if for every polynomially unpredictable $\mathcal{D}$ and for every $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2) \in \mathsf{FP}$:

$$\mathsf{Pr}[(d, a) \leftarrow \mathsf{A}_1(1^k), X \leftarrow \mathcal{D}, (c, r) \leftarrow \mathsf{A}_2(X, a) \colon \mathsf{Verify}(h(X), c, d) = \mathsf{yes} = \mathsf{Audit}(r, d)] = k^{-\omega(1)} \ . \tag{6}$$

Lemma 1 helps to prove the security of time-stamping schemes relative to a client-side hash function h.

Lemma 1. *If $\mathcal{D}$ is a polynomially unpredictable distribution on $\{0,1\}^*$ and $h \colon \{0,1\}^* \to \{0,1\}^k$ is a collision-resistant hash function then $h(\mathcal{D})$ is also polynomially unpredictable. (Appendix A)*

Note that a secure auditable time-stamping scheme (in the sense of (3) or (4)) is not necessarily secure relative to every collision-resistant hash function ((5),(6)) because, in the conditions (5) and (6) the adversary A_2 has more information (an h-pre-image X of x) than in (3) and (4).

4 Construction of a Provably Secure Auditable Time-Stamping Scheme

Let $h \colon \{0,1\}^* \to \{0,1\}^k$ be a collision-resistant hash function, or a universal one-way hash function chosen randomly by $\mathsf{Repository}$. We define $\mathsf{ATS}^h = (\mathsf{Publish}^h, \mathsf{Stamp}^h, \mathsf{Verify}^h, \mathsf{Report}^h, \mathsf{Audit}^h)$ and prove that this 5-tuple of functions form a secure time-stamping scheme with audit. Let $L = (x_0, \dots, x_{m-1})$ be all requests received during the t-th round. For simplicity, we assume that $m = 2^\ell$.

The publishing function $\mathsf{Publish}^h(L)$ builds a complete binary tree of height ℓ each vertex v of which has a $(k+1)$-bit label $\Lambda[v] = b\|H[v]$, where $b \in \{0,1\}$ indicates whether v is a leaf ($b = 0$ iff v is a leaf) and $H[v] \in \{0,1\}^k$ is a hash value computed by the following (inductive) scheme. For the n-th leaf v, we define $H[v] = x_n$, and $H[v] = h(\Lambda[v_L]\|\Lambda[v_R])$ for any non-leaf v, where v_L and v_R are the left- and the right child of v, respectively. As a result, $\mathsf{Publish}^h(L)$ returns a $(k+1)$-bit root label of the tree.

The stamping function $\mathsf{Stamp}^h(L,n)$ builds the same tree as above. Let v be the n-th leaf and $v = v_0, v_1, \ldots, v_{\ell-1}, v_\ell$ be the unique path from v to the root vertex (v_ℓ), i.e. v_i is a child of v_{i+1} for every $i \in \{0,\ldots,\ell-1\}$. Let $v'_0, v'_1, \ldots, v'_{\ell-1}$ denote the siblings of $v_0, v_1, \ldots, v_{\ell-1}$, respectively. Let $z_i = \Lambda[v'_i]$ for every $i \in \{0,\ldots,\ell-1\}$ and $z = (z_0,\ldots,z_{\ell-1})$. The certificate is $c = \mathsf{Stamp}^h(L,n) = (n,z)$.

The verification function $\mathsf{Verify}^h(x,(n,z),d)$ recomputes d (based on x and (n,z)) and compares the results. Let $n = n_{\ell'-1}n_{\ell'-2}\ldots n_0$ be the binary representation of n and $z = (z_0, z_1, \ldots, z_{\ell'-1})$. The verification function computes sequences $\lambda = (\lambda_0, \lambda_1, \ldots, \lambda_{\ell'}) \in \left(\{0,1\}^{k+1}\right)^{\ell'}$ and $\chi = (\chi_0, \chi_1, \ldots, \chi_{\ell'}) \in \left(\{0,1\}^k\right)^{\ell'}$ inductively, so that $\chi_0 = x$, $\lambda_0 := 0\|x$, and for every $i > 0$, $\lambda_i = 1\|\chi_i$, where

$$\chi_i := \begin{cases} h(z_i\|\lambda_{i-1}) \text{ if } n_{i-1} = 1 \\ h(\lambda_{i-1}\|z_i) \text{ if } n_{i-1} = 0 \end{cases} . \tag{7}$$

The verification procedure outputs yes, iff $\lambda_{\ell'} = d$.

The report function is trivial, i.e. $\mathsf{Report}^h(L) = L$ for every list L.

The audit function $\mathsf{Audit}^h(L,d)$ computes $d' = \mathsf{Publish}^h(L)$ and returns yes iff $d' = d$.

Lemma 2. **(A)** *If $x \notin L$, and $\mathsf{Audit}^h(L,d) = \mathsf{Verify}^h(x,c,d) = \mathsf{yes}$ then the h-calls of Verify^h and $\mathsf{Publish}^h$ contain a collision for h.* **(B)** *If $L \neq L'$ and $\mathsf{Publish}^h(L) = \mathsf{Publish}^h(L')$ then the h-calls performed by the $\mathsf{Publish}^h$ function contain a collision for h. (Appendix B)*

Theorem 1. *If h is a collision resistant hash function then a time-stamping scheme ATS^h with audit-supported publishing is secure relative to a client-side hash function h. (Appendix C)*

Theorem 2. *If h is a collision resistant hash function then a time-stamping scheme ATS^h with multi-round audit is secure relative to a client-side hash function h. (Appendix D)*

Lemma 2 directly implies the consistency condition (1) for ATS^h. Hence, we have proved that our construction is secure in the conventional sense.

Note also that it is probably *not possible* to prove that the scheme $\mathsf{TS}^h = (\mathsf{Publish}^h, \mathsf{Stamp}^h, \mathsf{Verify}^h)$ without audit is secure against random back-dating (2), based on the collision-resistance of h. The reason is that one can find an oracle $\mathcal{O}$ and choose a hash function h (that uses $\mathcal{O}$) so that h is collision-resistant but TS^h is still insecure [8]. As the ordinary reduction techniques relativize, the implication is that the security of TS^h cannot be proved (in ordinary way) in the real world either. In this sence, the audit functionality is crucial for provable security.

5 Universally Composable Time-Stamping Schemes

5.1 Universal Composability Framework

To prove that a cryptographic primitive is secure in *every reasonable application* the *universal composability* (UC) paradigm is used [1–4, 9–11, 21–23]. If the reader is not familiar with the UC paradigm, we recommend to study the seminal works by Canetti [10, 11] and the monograph on composability by Lindell [16]. Rather than using *ad hoc* behavior of adversaries, the UC paradigm defines an *ideal primitive* which is "obviously secure" and then proves that if $\mathsf{A} \in \mathsf{FP}$ is an adversary for an application of the real primitive then there is another adversary $\mathsf{A}' \in \mathsf{FP}$ for the same application in which the real primitive is replaced with the ideal primitive. Loosely speaking, no security incident in any application of the primitive is caused by the difference between the real and the ideal primitives. Hence, the real functionality faithfully implements the ideal functionality.

We use the language of Finite State Machine(FSM) theory borrowed from Pfitzmann [23] when describing the UC formalism. Every component of the system (for a fixed value of k) is a (probabilistic) FSM with input- and output ports. Each port has a name and a type (in or out). By a *composition* $\langle \mathsf{M}_1, \mathsf{M}_2 \rangle$ of two machines M_1 and M_2 we mean a network of machines obtained by connecting the input and output port pairs in a certain (pre-defined) way. For example, pairs with identical names can be connected. The precise formalism for describing the connections is unimportant in this paper, because the networks we use are very simple. We assume that each input port is buffered, whereas the length of the buffer is unlimited. When analyzing a particular machine, we use the following abbreviations. By $\mathsf{in}_\nu \to x$ we denote the event that the machine has input x in the port in_ν. By $y \to \mathsf{out}_\nu$ we mean that y is sent to the output port out_ν. To overcome the difficulties related to the asynchronous behavior of the network, it has been assumed that no two machines run at the same time. Technically, this condition is achieved by introducing the clock-ports to the system. Each machine, after finishing its work, can clock (give the token to) only one machine. In this paper, we use clocked output signals to represent the clocking. By $x \overset{\triangleright}{\to} \mathsf{out}_\nu$ we mean that x is sent to the output port named ν and the token is given to the machine with input port in_ν. By the *view* of M_i in a composition $\mathsf{M} = \langle \mathsf{M}_1, \ldots, \mathsf{M}_n \rangle$ we mean the sequence of all input/output signals of M_i in a particular run of M. The view is denoted by $\textsc{View}_{\mathsf{M}_i}\langle \mathsf{M}_1, \ldots, \mathsf{M}_n \rangle$. In general, the view is a probability space.

In the UC framework, we have an ideal time-stamping scheme TS_I, a real scheme TS_R, and an environment Client. A composition $\langle \mathsf{Client}, \mathsf{TS}_\mathrm{R} \rangle$ is called a *real application*, while $\langle \mathsf{Client}, \mathsf{TS}_\mathrm{I} \rangle$ is called an *ideal application*. Each machine has special input/output ports for an adversary A.

Definition 6 (Universal Composability). TS_R *is* universally composable, *if for every* $\mathsf{A} \in \mathsf{FP}$ *there is a* $\mathsf{A}' \in \mathsf{FP}$*, so that for every* $\mathsf{Client} \in \mathsf{FP}$*:* $\textsc{View}_{\mathsf{Client}}\langle \mathsf{Client}, \mathsf{TS}_\mathrm{R}, \mathsf{A} \rangle \approx \textsc{View}_{\mathsf{Client}}\langle \mathsf{Client}, \mathsf{TS}_\mathrm{I}, \mathsf{A}' \rangle$.

Informally, this condition mens that anything that can happen to the real application $\langle \mathsf{Client}, \mathsf{TS}_\mathrm{R} \rangle$ can also happen to the ideal application $\langle \mathsf{Client}, \mathsf{TS}_\mathrm{I} \rangle$.

In most of the proofs of universal composability, a simulator S is constructed that uses A as a black-box, i.e. $\mathsf{A}' = \langle \mathsf{S}, \mathsf{A} \rangle$. It is then proved that the behavior of $\langle \mathsf{TS}_\mathrm{R}, \mathsf{S} \rangle$ and TS_I are indistinguishable, except when certain cryptographic primitives (used by TS_R) are broken. Hence, if the primitives are believed to be secure, this implies the indistinguishability of views and also the security of TS_R in the strongest possible sense. To prove the identical behavior of $\langle \mathsf{TS}_\mathrm{R}, \mathsf{S} \rangle$ and TS_I, a *bisimulation* between these two machines is constructed.

5.2 On the Model

Some primitives are hard to cast in terms of the UC framework. The *commitment problem* occurs, meaning that a simulator that acts as an intermediary between the real-world adversary and the ideal functionality has to fix the value of a certain data item without knowing all the components it was created from, and also without the ability to present instead of this data item something that is and remains indistinguishable from it. Canetti and Fishlin proved [12] that UC bit-commitment is impossible in the in the "plain model" (i.e. a model without ideal functionalities) but it becomes possible in the Common Reference String model, where a common (and accessible) random string is added to the system as an ideal functionality. Similar problems occur when trying to define universally composable time-stamping schemes, but fortunately, the problems dissapear if an *ideal audit functionality* (represented in our model by $\mathsf{Repository}$ that is merged with $\mathsf{Auditor}$) is added to the system. The universal composability can be proved based on the *universal one-wayness* of a hash function h, assuming that a new random instance of h is generated (by $\mathsf{Repository}$) during each round. The reduction we obtain is linear-preserving and gives good practical security guarantees.

Hence, our UC Time-Stamping scheme construction is not in the plain model. However, adding the trusted $\mathsf{Repository}$ to the system is reasonable because: (1) there are real-life systems that behave in a similar way (e.g. newspapers); (2) it is possible to implement similar functionalities in the Common Reference String model by using public-key cryptography.

5.3 Ideal Time-Stamping Scheme

The ideal scheme is a secure host that stores for each round number t a set L_t of all bit-strings that were stamped during the t-th round. The value of t is initially 0 and is incremented each time the round is closed. In our real scheme, we allow the stamping functionality to be corrupted. This is reflected in the ideal scheme by giving the adversary complete control on which bit-strings will be considered stamped during the current round. As we shall see, at the end of the round t the adversary sends the contents of L_t to the secure host. Hence no availability is guaranteed. The important property is, however, that once the t-th round has ended, no more bit-strings can be added to L_t — *back-dating* is not possible.

In the real world, the verification of a time-stamp may fail for a number of reasons that are under the control of the adversary. For example, the repository may be currently unavailable or it may be available but not yet contain the digest of the round we are interested in. In this case we cannot rely on the time-stamp and must behave as if it was invalid. In the ideal world we model this situation by allowing the adversary to declare any verification attempt unsuccessful. However, the adversary is unable to declare a time-stamp valid if it really was not.

The internal state of the ideal time-stamping scheme TS_I consists of an indexed list $\mathfrak{L}_\mathrm{I}$ each element $\mathfrak{L}_\mathrm{I}[t]$ of which is a set of k-bit strings. Initially, $\mathfrak{L}_\mathrm{I} = [\,]$. The ideal scheme TS_I (Fig. 1, left) offers service on ports in_req, out_st, in_ver, and $\mathsf{out}_\mathsf{res}$. The other ports ($\mathsf{out}_\mathsf{req}$, in_st, in_aud, $\mathsf{out}_\mathsf{ver}$, and in_res) are intended for communication with an adversary A'. In the following, we describe the behavior of TS_I by defining its reaction to any possible input.

- If $\mathsf{in}_\mathsf{req} \to x$ then $x \to \mathsf{out}_\mathsf{req}$.
- If $\mathsf{in}_\mathsf{st} \to (x, t)$ then $(x, t) \to \mathsf{out}_\mathsf{req}$.
- If $\mathsf{in}_\mathsf{aud} \to L$ then $\mathfrak{L}_\mathrm{I} := \mathfrak{L}_\mathrm{I} \| L$.
- If $\mathsf{in}_\mathsf{ver} \to (x, \tau)$ then $(x, \tau) \to \mathsf{out}_\mathsf{ver}$.
- If $\mathsf{in}_\mathsf{res} \to (x, \bar{b}, \tau)$ then $b := \bar{b}\,\&\mathsf{True}(x \in \mathfrak{L}_\mathrm{I}[\tau])$ and $(x, b, \tau) \to \mathsf{out}_\mathsf{res}$.

5.4 Real Scheme

In the real scheme, the trusted host is replaced by a number of $\mathsf{Verifier}$ hosts. Some of them may be corrupted but we observe only one non-corrupted $\mathsf{Verifier}$. This is allowed because in the standard time-stamping setting, there is no communication between the verifiers. We assume that the channel between $\mathsf{Repository}$ and (non-corrupted) $\mathsf{Verifier}$ is tamperproof. It is a reasonable practical assumption because channels with similar security properties (e.g. newspapers) exist in the real life.

Having obtained a verification request (x, t) (which reads "Was x time-stamped during the t-th round?"), $\mathsf{Verifier}$ obtains the corresponding r_t from $\mathsf{Repository}$ and applies the Verify^h procedure. However, $\mathsf{Verifier}$ needs a certificate c for verification. We take into account possible (malicious) modification of the certificate before verification. Therefore, it is natural to assume that the certificate is entirely provided by the adversary A.

The real scheme TS_R (Fig. 2, right) consists of three components:

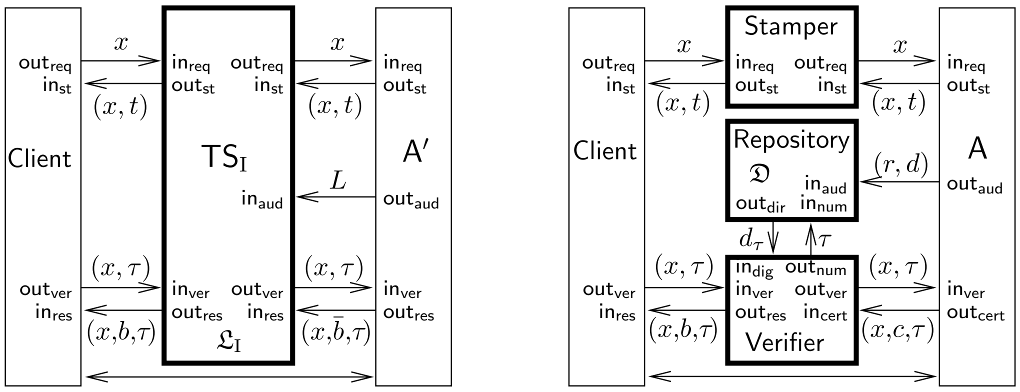

Fig. 2. The ideal scheme TS_I and the real scheme $\mathsf{TS}_R = \langle \mathsf{Stamper}, \mathsf{Repository}, \mathsf{Verifier} \rangle$

– Stamper – a prototype for a server that receives time stamp requests and returns time stamps to Client. As we assume that the adversary A has full control over Stamper, we define Stamper as a stateless intermediary between Client and A. Stamper offers service on ports $\mathsf{in}_{\mathsf{req}}$ and $\mathsf{out}_{\mathsf{st}}$. Two other ports ($\mathsf{out}_{\mathsf{req}}$ and $\mathsf{in}_{\mathsf{st}}$) are for the communication with A. The behavior of Stamper is defined as follows:
  - If $\mathsf{in}_{\mathsf{req}} \to x$ then $x \to \mathsf{out}_{\mathsf{req}}$.
  - If $\mathsf{in}_{\mathsf{st}} \to (x,t)$ then $(x,t) \to \mathsf{out}_{\mathsf{st}}$.

– Repository – a prototype for a secure repository that publishes the digests of rounds. The internal state of Repository consists of a (initially empty) list $\mathfrak{D}$ of k-bit strings. Repository offers service on ports $\mathsf{in}_{\mathsf{num}}$ and $\mathsf{out}_{\mathsf{dir}}$. The third port $\mathsf{in}_{\mathsf{aud}}$ is for the communication with A. The behavior of Repository is defined as follows:
  - If $\mathsf{in}_{\mathsf{num}} \to \tau$ and $\tau < |\mathfrak{D}|$, then $d_\tau := \mathfrak{D}[\tau]$ and $d_\tau \to \mathsf{out}_{\mathsf{dig}}$.
  - If $\mathsf{in}_{\mathsf{num}} \to \tau$ and $\tau \geq |\mathfrak{D}|$, then $\mathsf{NIL} \to \mathsf{out}_{\mathsf{dig}}$.
  - If $\mathsf{in}_{\mathsf{aud}} \to (r,d)$ and $\mathsf{Audit}(r,d) = \mathsf{yes}$ then $\mathfrak{D} := \mathfrak{D} \| d$.

– Verifier – a prototype for a real verification environment, which typically is a trusted client computer. Verifier receives verification requests and answers with a verification result. It is assumed that Verifier is able to obtain the digests d_τ form Repository in a tamperproof way. The internal state of Verifier consists of a bit-string variable $\mathfrak{r}$. Initially, $\mathfrak{r} = []$. Verifier offers service on ports $\mathsf{in}_{\mathsf{ver}}$ and $\mathsf{out}_{\mathsf{res}}$. Two ports – $\mathsf{out}_{\mathsf{num}}$ and $\mathsf{in}_{\mathsf{dig}}$ – are for requesting the digests from Repository, and two last ports ($\mathsf{out}_{\mathsf{ver}}$ and $\mathsf{in}_{\mathsf{dig}}$) are for the communication with A. Let $y \overset{\triangleright}{\to} \mathsf{out}_{\mathsf{c}}$ denote the event that y is sent to the output channel $\mathsf{out}_{\mathsf{c}}$ and the corresponding connection is clocked. The behavior of Verifier is defined as follows:
  - If $\mathsf{in}_{\mathsf{ver}} \to (x,\tau)$ then $(x,\tau) \to \mathsf{out}_{\mathsf{ver}}$.
  - If $\mathsf{in}_{\mathsf{cert}} \to (x,c,\tau)$ then $\mathfrak{r} := (x,c,\tau)$ and $\tau \overset{\triangleright}{\to} \mathsf{out}_{\mathsf{num}}$.
  - If $\mathsf{in}_{\mathsf{dig}} \to d_\tau \in \{0,1\}^k$ and $\mathfrak{r} = (x,c,\tau)$, then $b := \mathsf{Verify}(x,c,d_\tau)$ and $(x,b,\tau) \to \mathsf{out}_{\mathsf{res}}$.
  - If $\mathsf{in}_{\mathsf{dig}} \to \mathsf{NIL}$ and $\mathfrak{r} = (x,c,\tau)$, then $(x,\mathsf{no},\tau) \to \mathsf{out}_{\mathsf{res}}$.

For completing the description of the real scheme, it is sufficient to give efficient constructions for Publish, Audit, and Verify, i.e. exactly the components of an auditable time-scheme that appear in the security conditions (3), (4), (5), and (6). Hence, for any auditable time-stamping scheme it is reasonable to speak about *universal composability* as a security condition. In the next subsection, we define a simulator for the scheme ATS^h, we presented in Section 4.

5.5 Simulator for ATS^h

The internal state of the simulator S (Fig. 2, right) consists of two lists $(\mathfrak{D}_\mathrm{I}, \mathfrak{C}_\mathrm{I})$ and a bit-string $\mathfrak{r}_\mathrm{I}$. The elements of $\mathfrak{D}_\mathrm{I}$ are k-bit strings, while the elements of $\mathfrak{C}_\mathrm{I}$ are sets of k-bit strings. Initially, $\mathfrak{D}_\mathrm{I} = \lfloor\rfloor$, $\mathfrak{C}_\mathrm{I} = (\emptyset, \emptyset, \ldots)$, and $\mathfrak{r}_\mathrm{I} = \lfloor\rfloor$. The simulator has five ports (in_req, out_st, $\mathsf{out}_\mathsf{aud}$, in_ver, and $\mathsf{out}_\mathsf{res}$) for communicating with TS_I and five ports ($\mathsf{out}_\mathsf{req}$, in_st, in_aud, $\mathsf{out}_\mathsf{ver}$, and $\mathsf{in}_\mathsf{cert}$) for communicating with A. The behavior of S is defined as follows:

- If $\mathsf{in}_\mathsf{req} \rightarrow x$ then $x \rightarrow \mathsf{out}_\mathsf{req}$.
- If $\mathsf{in}_\mathsf{st} \rightarrow (x, t)$ then $(x, t) \overset{\triangleright}{\rightarrow} \mathsf{out}_\mathsf{st}$.
- If $\mathsf{in}_\mathsf{aud} \rightarrow (L, d)$ and $d = \mathsf{Publish}(L)$, then $\mathfrak{D}_\mathrm{I} := \mathfrak{D}_\mathrm{I} \| d$, and $L \overset{\triangleright}{\rightarrow} \mathsf{out}_\mathsf{aud}$.
- If $\mathsf{in}_\mathsf{ver} \rightarrow (x, \tau)$ then $(x, \tau) \rightarrow \mathsf{out}_\mathsf{ver}$.
- If $\mathsf{in}_\mathsf{cert} \rightarrow (x, c, \tau)$ then $\mathfrak{r}_\mathrm{I} := (x, c, \tau)$, $\bar{b} := \tau < |\mathfrak{D}_\mathrm{I}|\ \&\, \mathsf{Verify}(x, c, \mathfrak{D}_\mathrm{I}[\tau])$, and $(x, \bar{b}, \tau) \overset{\triangleright}{\rightarrow} \mathsf{out}_\mathsf{res}$. If $\bar{b} = \mathsf{yes}$ then $\mathfrak{C}_\mathrm{I}[\tau] := \mathfrak{C}_\mathrm{I}[\tau] \cup \{x\}$.

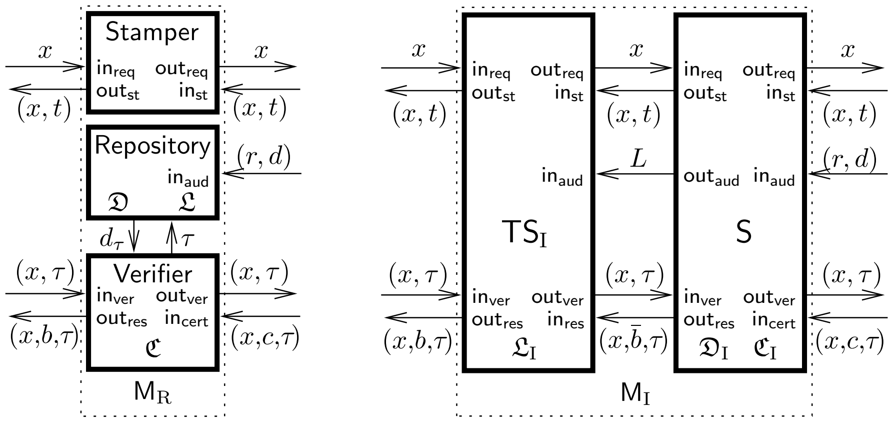
Stamper
x
in_req $\mathsf{out}_\mathsf{req}$
out_st in_st
(x, t)
Repository
(r, d)
in_aud
$\mathfrak{D}$ $\mathfrak{L}$
d_τ τ
Verifier
(x, τ)
in_ver $\mathsf{out}_\mathsf{ver}$
$\mathsf{out}_\mathsf{res}$ $\mathsf{in}_\mathsf{cert}$
(x,b,τ)
(x,c,τ)
$\mathfrak{C}$
M_R
TS_I
L
$\mathsf{out}_\mathsf{aud}$ in_aud
S
in_res
$(x,\bar{b},\tau)$
$\mathfrak{L}_\mathrm{I}$
$\mathfrak{D}_\mathrm{I}$ $\mathfrak{C}_\mathrm{I}$
M_I

Fig. 3. The real machine M_R, the simulator S, and the ideal machine $\mathsf{M}_\mathrm{I} = \langle \mathsf{TS}_\mathrm{I}, \mathsf{S}\rangle$

5.6 Bisimilarity of the Real- and the Ideal Machines

We start the proof by augmenting the state of the components Repository and Verifier of the real functionality. From the following description it is obvious that this extra state does not influence the behavior of these components as the existing parts make no use of the new state.

We add an initially empty list $\mathfrak{L}$ of sets of k-bit strings to the state of Repository. We also replace the third item in the description of its behavior by

- If $\mathsf{in}_\mathsf{aud} \rightarrow (r, d)$ and $\mathsf{Audit}(r, d) = \mathsf{yes}$ then $\mathfrak{D} := \mathfrak{D} \| d$, and $\mathfrak{L} := \mathfrak{L} \| r$.

We add a list $\mathfrak{C}$ of sets of k-bit strings to the state of Verifier. Initially, $\mathfrak{C}$ is an infinite list with all elements equal to $\emptyset$. We replace the third item in the description of the behavior of Verifier by

- If $\mathsf{in}_\mathsf{dig} \rightarrow d_\tau \in \{0,1\}^k$ and $\mathfrak{r} = (x, c, \tau)$, then $b := \mathsf{Verify}(x, c, d_\tau)$ and $(x, b, \tau) \rightarrow \mathsf{out}_\mathsf{res}$. If $b = \mathsf{yes}$ then $\mathfrak{C}[\tau] := \mathfrak{C}[\tau] \cup \{x\}$.

Let $\mathsf{M}_\mathrm{R} = \mathsf{TS}_\mathrm{R}$ be the real machine and $\mathsf{M}_\mathrm{I} = \langle \mathsf{TS}_\mathrm{I}, \mathsf{S}\rangle$ be the ideal machine (Fig. 2). A state $s = (\mathfrak{L}, \mathfrak{D}, \mathfrak{C}, \mathfrak{r})$ is said to be *faulty* if $\exists \tau\colon \mathfrak{C}[\tau] \not\subseteq \mathfrak{L}[\tau]$. Let S_R and S_I be the sets of states of M_R and M_I, respectively. Let F_R and F_I be the corresponding sets of faulty states. Let I and O be the sets of inputs and outputs (common for M_{R} and M_{I}). Let $\delta_{\mathrm{R}}\colon I \times S_{\mathrm{R}} \to S_{\mathrm{R}}$ be the next-state function of M_{R} and $\lambda_{\mathrm{R}}\colon I \times S_{\mathrm{R}} \to O$ be the output function of M_{R}. We define δ_{I} and λ_{I} analogously for M_{I}. Let s^0_{R} and s^0_{I} be the initial states of the corresponding machines.

It is easy to verify (by using Lemma 2) that if one of the machines reaches a faulty state then the h-calls performed so far comprise a collision for h.

Definition 7. *Two machines* M_{R} *and* M_{I} *are said to be* bisimilar with error $(F_{\mathrm{R}}, F_{\mathrm{I}})$*, if there is a binary relation (called a* bisimulation*)* $\beta \subseteq S_{\mathrm{R}} \times S_{\mathrm{I}}$ *such that* $(s^0_{\mathrm{R}}, s^0_{\mathrm{I}}) \in \beta$ *and for any pair of states* $(s_{\mathrm{R}}, s_{\mathrm{I}}) \in \beta$ *and for any input* $i \in I$*, at least one of the following three conditions holds:* **(1)** $\delta_{\mathrm{R}}(i, s_{\mathrm{R}}) \in F_{\mathrm{R}}$*,* **(2)** $\delta_{\mathrm{I}}(i, s_{\mathrm{I}}) \in F_{\mathrm{I}}$*, or* **(3)** $(\delta_{\mathrm{R}}(i, s_{\mathrm{R}}), \delta_{\mathrm{I}}(i, s_{\mathrm{I}})) \in \beta$ *and* $\lambda_{\mathrm{R}}(i, s_{\mathrm{R}}) = \lambda_{\mathrm{I}}(i, s_{\mathrm{I}})$*.*

Theorem 3. *(See Appendix E for a proof) The machines* $\mathsf{M}_{\mathrm{R}} = \mathsf{TS}_{\mathrm{R}}$ *and* $\mathsf{M}_{\mathrm{I}} = \langle \mathsf{TS}_{\mathrm{I}}, \mathsf{S} \rangle$ *are bisimilar with error* $(F_{\mathrm{R}}, F_{\mathrm{I}})$*, whereas the bisimulation* β *is defined as follows:*

$$(\mathfrak{L}, \mathfrak{D}, \mathfrak{C}, \mathfrak{r})\, \beta\, (\mathfrak{L}_{\mathrm{I}}, \mathfrak{D}_{\mathrm{I}}, \mathfrak{C}_{\mathrm{I}}, \mathfrak{r}_{\mathrm{I}}) \quad \equiv \quad (\mathfrak{L} = \mathfrak{L}_{\mathrm{I}})\&(\mathfrak{D} = \mathfrak{D}_{\mathrm{I}})\&(\mathfrak{C} = \mathfrak{C}_{\mathrm{I}})\&(\mathfrak{r} = \mathfrak{r}_{\mathrm{I}}).$$

Corollary 1. *If* h *is a collision-resistant (or universal one-way) hash function then* ATS^h *is a universally composable time-stamping scheme with audit-supported publishing.*

6 Size of the Audit Report

In ATS^h, the Report function is not length-decreasing which means that the network load (and the computations) are doubled compared to the schemes without audit. It is natural to ask whether the length of the report can be reduced. The answer turns out to be negative: in every universally composable time-stamping scheme with audit-supported publishing $|r_t| \approx |L_t|$.

We construct a Client and an adversary A (for $\langle \mathsf{Client}, \mathsf{TS}_{\mathrm{R}} \rangle$) so that no efficient adversary A′ (for $\langle \mathsf{Client}, \mathsf{TS}_{\mathrm{I}} \rangle$) can simulate A unless the length of $\mathsf{Report}(r_t)$ is almost $|L_t|$. Our construction exploits the commitment problem – the adversary A′ (or a simulator) knows only d_t but has to send L_t to TS_{I}, and hence L_t should be efficiently computable from d_t.

The internal state of Client consists of a $p(k)$-element array $L = (x_1, \ldots, x_{p(k)}) \in \{0,1\}^{k \times p(k)}$ (where $p(k) = k^{O(1)}$), a k-bit string z (initially 0), and a boolean value Roundover that is initially false. Client reacts to the input events as follows:

- If $\mathsf{in}_{\mathsf{A}} \to \mathsf{init}$ then the Client generates $x_1, \ldots, x_{p(k)}$ independently at random, computes $r = \mathsf{Report}(L)$, $d = \mathsf{Publish}(L)$, and outputs $(r, d) \to \mathsf{out}_{\mathsf{A}}$.
- If $\mathsf{in}_{\mathsf{A}} \to \mathsf{round}$ then the Client outputs $(0^k, 1) \overset{\triangleright}{\to} \mathsf{out}_{\mathsf{ver}}$.
- If $\mathsf{in}_{\mathsf{ver}} \to (0^k, \mathsf{yes}, 1)$ (a confirmation that the round is closed) then the Client sets Roundover $:=$ true and outputs $L \to \mathsf{out}_{\mathsf{A}}$ (reveals L to the adversary).
- If $\mathsf{in}_{\mathsf{A}} \to \mathsf{verify}$ then Client generates $i \leftarrow \{1, \ldots, p(k)\}$ uniformly at random, sets $z := x_i$ and outputs $(z, 0) \overset{\triangleright}{\to} \mathsf{out}_{\mathsf{ver}}$.
- If $\mathsf{in}_{\mathsf{ver}} \to (z, \mathsf{yes}, 0)$ and Roundover $=$ yes then Client outputs yes $\to \mathsf{out}_{\mathsf{A}}$.

The adversary A is defined as follows. The internal state of A consists of a $p(k)$-element array $L_{\mathsf{A}} = (a_1, \ldots, a_{p(k)}) \in \{0,1\}^{k \times p(k)}$ (where $p(k) = k^{O(1)}$). First of all, the adversary A outputs $\mathsf{init} \to \mathsf{out}_{\mathsf{Client}}$ and then reacts to the input events as follows:

- If $\mathsf{in}_{\mathsf{Client}} \to (r, d)$ then A outputs $(r, d) \overset{\triangleright}{\to} \mathsf{out}_{\mathsf{aud}}$. After getting control again, the adversary A outputs $(\mathsf{Report}(0^k), \mathsf{Publish}(0^k)) \overset{\triangleright}{\to} \mathsf{out}_{\mathsf{aud}}$. Finally, A outputs round $\overset{\triangleright}{\to} \mathsf{out}_{\mathsf{Client}}$.
- If $\mathsf{in}_{\mathsf{ver}} \to (0^k, 1)$ then A outputs $(0^k, \mathsf{Stamp}(0^k, 1), 0) \to \mathsf{out}_{\mathsf{cert}}$.
- If $\mathsf{in}_{\mathsf{Client}} \to L$ then A sets $L_{\mathsf{A}} := L$ and outputs verify $\overset{\triangleright}{\to} \mathsf{out}_{\mathsf{Client}}$.

– If $\mathsf{in}_{\mathsf{ver}} \to (z, 0)$ then A finds an i, such that $L_{\mathsf{A}} = z$, computes $c := \mathsf{Stamp}(i, L_{\mathsf{A}})$, and outputs $(z, c, 0) \to \mathsf{out}_{\mathsf{cert}}$. The adversary halts if there is no such i.

It is easy to see that with probability one, the client's view $\mathrm{VIEW}_{\mathsf{Client}}\langle \mathsf{Client}, \mathsf{TS}_{\mathrm{R}}, \mathsf{A}\rangle$ contains the output yes from Client. Let $\mathsf{A}' \in \mathsf{FP}$ and $\mathrm{VIEW}_{\mathsf{Client}}\langle \mathsf{Client}, \mathsf{TS}_{\mathrm{R}}, \mathsf{A}\rangle \approx \mathrm{VIEW}_{\mathsf{Client}}\langle \mathsf{Client}, \mathsf{TS}_{\mathrm{I}}, \mathsf{A}'\rangle$. Hence, due to the indistinguishability, with probability $1 - k^{-\omega(1)}$ the view $\mathrm{VIEW}_{\mathsf{Client}}\langle \mathsf{Client}, \mathsf{TS}_{\mathrm{I}}, \mathsf{A}'\rangle$ contains the output yes from Client. From the description of TS_{I}, it follows that with probability $1 - k^{-\omega(1)}$ the adversary A' (based on partial information $(\mathsf{Report}(L), \mathsf{Publish}(L))$) is capable of finding L_{A} such that $x_i \in L_{\mathsf{A}}$. Lemma 3 below shows that such A' is possible only if the bit-length of $(\mathsf{Report}(L), \mathsf{Publish}(L))$ is $\approx k \cdot p(k)$.

Lemma 3. *Let* $X_1, \ldots, X_{p(k)} \in \{0,1\}^k$ *(*$p(k) = k^{O(1)}$*) and* $\Im \leftarrow \{1, \ldots, p(k)\}$ *be independent uniformly distributed random variables. Let* $f\colon \{0,1\}^{k \times p(k)} \to \{0,1\}^{\ell(k)}$ *and* $A\colon \{0,1\}^{\ell(k)} \to \{0,1\}^{k \times m(k)}$ *(where* $m(k) = k^{O(1)}$*) be function families, such that*

$$\delta = \Pr[X_1, \ldots, X_{p(k)} \leftarrow \{0,1\}^k, L \leftarrow A(f(X_1, \ldots, X_{p(k)})), \Im \leftarrow \{1, \ldots, p(k)\}\colon X_{\Im} \in L] = 1 - k^{-\omega(1)}.$$

Then $\ell(k) = k \cdot p(k) - O(\log k)$.

Proof. A $p(k)$-tuple $(x_1, \ldots, x_{p(k)})$ is *good* if $x_i \in A(f(x_1, \ldots, x_{p(k)}))$ for all $i \in \{1, \ldots, p(k)\}$. Other tuples are *bad*. As for any bad tuple $(x_1, \ldots, x_{p(k)})$ the probability of error $1 - \delta \geq \frac{1}{p(k)} \neq k^{-\omega(1)}$, the number of good tuples should be $(1 - k^{-\omega(1)}) \cdot 2^{k \cdot p(k)}$. On the other hand, the number of good tuples cannot exceed $2^{\ell(k)} \cdot m(k)^{p(k)}$ and hence $2^{\ell(k)} \cdot m(k)^{p(k)} = (1 - k^{-\omega(1)}) \cdot 2^{k \cdot p(k)}$, which gives (by taking logarithm from both sides) $\ell(k) = k \cdot p(k) - O(\log k)$. □

Corollary 2. *In every universally composable time-stamping scheme with audit-supported publishing* $\mathsf{ATS} = (\mathsf{Publish}, \mathsf{Stamp}, \mathsf{Verify}, \mathsf{Report}, \mathsf{Audit})$*, where the report and the publishing functions have types:* $\mathsf{Report}\colon \{0,1\}^{k \times p(k)} \to \{0,1\}^{r(k)}$ *and* $\mathsf{Publish}\colon \{0,1\}^{k \times p(k)} \to \{0,1\}^{d(k)}$*:*

$$r(k) + d(k) \geq k \cdot p(k) - O(\log k),$$

i.e. the amount of information sent to the auditor is comparable to the list of all time stamps.

Actually, the last corollary holds for a weaker security notion that is called *simulatability*:

Definition 8 (Simulatability). *The real scheme* TS_{R} *is* simulatable, *if for every* $\mathsf{Client}, \mathsf{A} \in \mathsf{FP}$ *there is* $\mathsf{A}' \in \mathsf{FP}$*, so that* $\mathrm{VIEW}_{\mathsf{Client}}\langle \mathsf{Client}, \mathsf{TS}_{\mathrm{R}}, \mathsf{A}\rangle \approx \mathrm{VIEW}_{\mathsf{Client}}\langle \mathsf{Client}, \mathsf{TS}_{\mathrm{I}}, \mathsf{A}'\rangle$.

Like the Universal Composability, also the Simulatability implies both the Consistency (1) and the security against Random Back-Dating (2) but not the other way round. So it is still possible that one can compress the published information and still have a provably secure auditable time-stamping scheme in the sense of (1) and (2). One such construction is presented in [8] but their security reduction is very inefficient.

The Simulatability (and the Universal Composability) conditions depend on the definitions of TS_{I} and TS_{R}. It is not completely excluded that these definitions can be relaxed (in a reasonable way) so that the compression of the published information becomes possible. However, we cannot even imagine how this could be done.

7 Discussion on Practical Implementation

As in the schemes with audit, all time stamp requests are sent from Stamper to Auditor who then performs the same hash computations. Hence, if there are m stampers in the scheme and each stamper performs p hash operations per round, then the auditor must perform $m \cdot p$ hash operations per round. Hence, the cost of the service increases by a constant factor, no matter how many users there are.

In the schemes described above, we have only one trusted $\mathsf{Auditor}$. As one of our main goal was to develop measures against insider attacks, it is reasonable to assume that also the $\mathsf{Auditor}$ can be malicious. A natural approach would be to replace a trusted $\mathsf{Auditor}$ with a list $\mathsf{Auditor}_1, \ldots, \mathsf{Auditor}_n$ of auditors and use secure multi-party computation. A simplified approach would be that a $\mathsf{Stamper}$ sends the digest d and the report r to all auditors in the list. The auditors check whether $\mathsf{Audit}(r, d) = 1$ and send r, d and the result (of the check) to $\mathsf{Repository}$ who then decides by clear majority which value to publish. This solution works, whenever the $\mathsf{Repository}$ and $\frac{n+1}{2}$ of the auditors are honest.

References

1. Michael Backes. *Cryptographically Sound Analysis of Security Protocols*. PhD thesis, Universität des Saarlandes, 2002.
2. Michael Backes and Birgit Pfitzmann. Symmetric Encryption in a Simulatable Dolev-Yao Style Cryptographic Library. In *17th IEEE Computer Security Foundations Workshop*, Pacific Grove, CA, June 2004.
3. Michael Backes, Birgit Pfitzmann, and Michael Waidner. Symmetric authentication within a simulatable cryptographic library. In Einar Snekkenes and Dieter Gollmann, editors, *Computer Security - ESORICS 2003, 8th European Symposium on Research in Computer Security*, volume 2808 of *LNCS*, pages 271–290, Gjøvik, Norway, October 2003. Springer-Verlag.
4. Michael Backes, Birgit Pfitzmann, and Michael Waidner. A Universally Composable Cryptographic Library. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, Washington, DC, October 2003. ACM Press.
5. Dave Bayer, Stuart Haber, and W.-Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences II: Methods in Communication, Security, and Computer Science*, pp.329-334, Springer-Verlag, New York 1993.
6. Josh Benaloh and Michael de Mare. Efficient broadcast time-stamping. Tech. report 1, Clarkson Univ. Dep. of Mathematics and Computer Science, August 1991.
7. Ahto Buldas, Peeter Laud, Helger Lipmaa, and Jan Villemson. Time-Stamping with Binary Linking Schemes. In *Advances in Cryptology – CRYPTO'98, LNCS* 1462, pp. 486-501, 1998.
8. Ahto Buldas and Märt Saarepera. On provably secure time-stamping schemes. In *Advances in Cryptology – ASIACRYPT 2004, LNCS 3329*, pp.500–514, 2004.
9. Ran Canetti. A unified framework for analyzing security of protocols. *Electronic Colloquium on Computational Complexity (ECCC)*, 8(16), 2001.
10. Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1):143-202, 2000.
11. Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In 42*nd FOCS*, pp. 136–145. 2001.
12. Ran Canetti and Marc Fischlin. Universally Composable Commitments. In *CRYPTO'01*, LNCS 2139, pp. 19–40. 2001.
13. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
14. Stuart Haber and W.-Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, Vol. 3, No. 2, pp. 99-111 (1991).
15. Stuart Haber and W.-Scott Stornetta. Secure Names for Bit-Strings. In *ACM Conference on Computer and Communications Security*, pp. 28–35, 1997.
16. Yehuda Lindell. *Composition of Secure Multi-Party Protocols*. A Comprehensive Study. LNCS 2815. 2003.
17. Michael Luby. *Pseudorandomness and cryptographic applications*. Princeton University Press, 1996.
18. Ralph C. Merkle. Protocols for public-key cryptosystems. *Proceedings of the 1980 IEEE Symposium on Security and Privacy*, pp.122-134, 1980.
19. Tal Moran, Ronen Shaltiel and Amnon Ta-Shma. Non-interactive timestamping in the bounded storage model. In *Advances in Cryptology – CRYPTO 2004, LNCS* 3152, 2004.
20. Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. Proceedings of the *Twenty First Annual ACM Symposium on Theory of Computing*. May 15–17 1989: Seattle, ACM Press, pp. 33–43, 1989.
21. Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Cryptographic Security of Reactive Systems. In Steve Schneider and Peter Ryan, editors, *Workshop on Secure Architectures and Information Flow*, volume 32 of *Electronic Notes in Theoretical Computer Science*, Royal Holloway, University of London, 2000. Elsevier Science.
22. Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *CCS 2000, Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 245–254, Athens, Greece, November 2000. ACM Press.

23. Birgit Pfitzmann and Michael Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In *2001 IEEE Symposium on Security and Privacy*, pages 184–200, Oakland, California, May 2001. IEEE Computer Society Press.
24. Alexander Russell. Necessary and sufficient conditions for collision-free hashing. *Journal of Cryptology* (1995) 8: 87–99.
25. `www.surety.com`
26. `www.authentidate.com`
27. `www.digistamp.com`

A Proof of Lemma 1

Let $\mathsf{\Pi}$ be a predictor with success probability $\delta(k) = \mathsf{Pr}[L \leftarrow \mathsf{\Pi}(1^k), X \leftarrow \mathcal{D}\colon h(X) \in L]$. Consider the following adversary $\mathsf{\Gamma}$ that on input 1^k:

– Calls $\mathsf{\Pi}$ to obtain $L \leftarrow \mathsf{\Pi}(1^k)$.
– Generates two independent bit-strings $X_1 \leftarrow \mathcal{D}$ and $X_2 \leftarrow \mathcal{D}$ and outputs (X_1, X_2).

Let $p(L_0) = \mathsf{Pr}[L \leftarrow \mathsf{\Pi}'(1^k)\colon L = L_0]$ and let E be the event that $h(X_1) = h(X_2) \in L_0$. Hence,

$$\begin{aligned}
\mathsf{Pr}[\mathsf{E}] &= \sum_{L_0} p(L_0)\cdot \mathsf{Pr}[X_1, X_2 \leftarrow \mathcal{D}\colon h(X_1) = h(X_2) \in L_0] \\
&= \sum_{L_0} p(L_0)\cdot \sum_{\ell\in L_0} \mathsf{Pr}[X_1, X_2 \leftarrow \mathcal{D}\colon h(X_1) = h(X_2) = \ell] = \sum_{L_0} p(L_0)\cdot \sum_{\ell\in L_0} \mathsf{Pr}^2[X \leftarrow \mathcal{D}\colon h(X) = \ell] \\
&= \sum_{L_0} p(L_0)\cdot \mathsf{Pr}^2[X \leftarrow \mathcal{D}\colon h(X) \in L_0] \cdot \sum_{\ell\in L_0} \mathsf{Pr}^2[h(X) = \ell \mid h(X) \in L_0] \\
&\geq \sum_{L_0} p(L_0)\cdot \mathsf{Pr}^2[X \leftarrow \mathcal{D}\colon h(X) \in L_0] \cdot \frac{1}{|L_0|} \geq \frac{1}{p(k)} \cdot \sum_{L_0} p(L_0)\cdot \mathsf{Pr}^2[X \leftarrow \mathcal{D}\colon h(X) \in L_0] \\
&\geq \frac{1}{p(k)} \cdot \left(\sum_{L_0} p(L_0)\cdot \mathsf{Pr}[X \leftarrow \mathcal{D}\colon h(X) \in L_0] \right)^2 = \frac{\delta^2(k)}{T(k)},
\end{aligned}$$

where $T(k)$ denotes the running time of $\mathsf{\Pi}(1^k)$. Note that if $X_1 \neq X_2$ then the event E leads to a collision of h. As $\mathsf{Pr}[X_1 \neq X_2] = 1 - k^{-\omega(1)}$ (otherwise, $\mathcal{D}$ would be a successful predictor for itself), then we conclude that $\mathsf{\Gamma}$ finds collisions with time-success ratio $\frac{T^2(k)}{\delta^2(k)} - k^{-\omega(1)}$. □

B Proof of Lemma 2.

(A) Let $c = (n, z, t)$. Let ℓ be the height of the tree and ℓ' be the bit-length of n. We prove the lemma by induction on $\ell + \ell'$. As a basis of the induction, we study the special cases $\ell = 0$ and $\ell' = 0$ and then perform the induction step, which states that if the statement holds in the case $\ell - 1$ and $\ell' - 1$ then it holds for ℓ and ℓ' as well.

Let $\ell' = 0$. In this case, $z = ()$ is an empty list and hence $\lambda_{\ell'} = 0\|x$, which is possible only if there is only one vertex in the tree, because otherwise the first bit of the root hash is 1. It follows that $x \in L$ because the the leaves of the tree are in one-to-one relationship with the elements of L. As we have a contradiction, the statement of the lemma trivially holds.

Let $\ell = 0$. In this case, the tree consists of a single vertex labeled with $\lambda_{\ell'} = 0\|\chi_{\ell'}$. But (by the definition of Verify^h), $\lambda_{\ell'}$ cannot begin with a 0-bit except if $\ell' = 0$, which contradicts $x \notin L$.

Now assume that ℓ' and ℓ are both positive and the statement holds for the smaller values of $\ell + \ell'$. As $\mathsf{Audit}^h(L, d) = \mathsf{yes}$, we have $\mathsf{Publish}^h(L) = d$. Therefore, the label of the loot vertex v is $y = 1\|h(\Lambda[v_L]\|\Lambda[v_R])$ (computed by $\mathsf{Publish}^h(L)$) is equal to $\lambda_{\ell'} \in \{h(z_{\ell'}\|\lambda_{\ell'-1}), h(\lambda_{\ell'-1}\|z_{\ell'})\}$ (computed by Verify^h). If $\Lambda[v_L]\|\Lambda[v_R] \notin \{z_{\ell'}\|\lambda_{\ell'-1}, \lambda_{\ell'-1}\|z_{\ell'}\}$ then we have a collision.

Otherwise, we assume without loss of generality that $\Lambda[v_L] = z_{\ell'}$ and $\Lambda[v_R] = \lambda_{\ell'-1}$. Let L_R be the set of all leaves in the right subtree of v. Note that the root label of the right subtree is $\overline{d} = \Lambda[v_R]$. Hence, we have $\mathsf{Publish}^h(L_R) = \lambda_{\ell'-1}$. Let $\overline{z} = (z_0, \ldots, z_{\ell'-2})$ and $\overline{n} = n_{\ell'-2} \ldots n_0$. Hence, we have $\mathsf{Audit}^h(L_R, \overline{d}) = \mathsf{Verify}^h(x, (\overline{n}, \overline{z}, t), \overline{d}) = \mathsf{yes}$. The statement (A) follows.
(B) As $L \neq L'$ there is an x which belong to one and only one of the lists L, L'. Assume without loss of generality that $x \in L$ but $x \notin L'$. Let n be the sequence number of x in L. Now simulate $c = \mathsf{Stamp}^h(L, n)$. Due to the correctness condition, we have $\mathsf{Verify}^h(x, c, \mathsf{Publish}^h(L)) = \mathsf{yes} = \mathsf{Audit}^h(L', \mathsf{Publish}^h(L))$, which by (A) leads to a collision. □

C Proof of Theorem 1

Let $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ be an adversary with success probability

$$\delta(k) = \Pr[(d, L, a) \leftarrow \mathsf{A}_1(1^k), X \leftarrow \mathcal{D}, c \leftarrow \mathsf{A}_2(X, a) \colon \mathsf{Verify}^h(h(X), c, d) = \mathsf{yes} = \mathsf{Audit}^h(L, d)] \enspace .$$

We define two adversaries: a *collision-finder* Γ for h and a *predictor* Π that predicts $h(\mathcal{D})$.

- *The collision finder* $\Gamma(1^k)$ simulates $(d, L, a) \leftarrow \mathsf{A}_1(1^k), X \leftarrow \mathcal{D}, c \leftarrow \mathsf{A}_2(X, a)$.
  If $\mathsf{Verify}^h(h(X), c, d) = \mathsf{yes} = \mathsf{Audit}^h(L, d)$ and $h(X) \notin L$ then Γ finds a collision by Lemma 2.
- *The predictor* $\Pi(1^k)$ simulates $(d, L, a) \leftarrow \mathsf{A}_1(1^k)$ and outputs L.

Considering an experiment $(d, L, a) \leftarrow \mathsf{A}_1(1^k), X \leftarrow \mathcal{D}, c \leftarrow \mathsf{A}_2(X, a)$, we define the following events:

$$\begin{aligned} \mathsf{Succ} &= [\mathsf{Verify}(h(X), c, d) = \mathsf{yes} = \mathsf{Audit}^h(L, d)] \\ \mathsf{Coll} &= [\mathsf{Verify}(h(X), c, d) = \mathsf{yes} = \mathsf{Audit}^h(L, d)] \cap [h(X) \notin L] \\ \mathsf{Pred} &= [\mathsf{Verify}(h(X), c, d) = \mathsf{yes} = \mathsf{Audit}^h(L, d)] \cap [h(X) \in L] \enspace . \end{aligned}$$

Let $\gamma(k)$ and $\pi(k)$ denote the success of Γ and Π, respectively. As $\mathsf{Succ} = \mathsf{Coll} \cup \mathsf{Pred}$, we have

$$\gamma(k) + \pi(k) \geq \Pr[\mathsf{Coll}] + \Pr[\mathsf{Pred}] \geq \Pr[\mathsf{Coll} \cap \mathsf{Pred}] = \Pr[\mathsf{Succ}] = \delta(k) \enspace .$$

As $\pi(k) = k^{-\omega(1)}$ by Lemma 1, it follows that Γ finds collisions with non-negligible success. □

D Proof of Theorem 2

Let $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ be an adversary with success probability

$$\delta(k) = \Pr[(d, a) \leftarrow \mathsf{A}_1(1^k), X \leftarrow \mathcal{D}, (c, L) \leftarrow \mathsf{A}_2(X, a) \colon \mathsf{Verify}^h(h(X), c, d) = \mathsf{Audit}^h(L, d) = \mathsf{yes}] \enspace .$$

Consider an experiment with the following computations:

$$(d, a) \leftarrow A_1(1^k), \begin{cases} X \leftarrow \mathcal{D}, & (c, L) \leftarrow \mathsf{A}_2(X, a), \quad v \leftarrow \mathsf{Verify}^h(h(X), c, d), \quad w \leftarrow \mathsf{Audit}^h(L, d) \\ X' \leftarrow \mathcal{D}, & (c', L') \leftarrow A_2(X', a), v' \leftarrow \mathsf{Verify}^h(h(X'), c', d), w' \leftarrow \mathsf{Audit}^h(L', d) \end{cases} \enspace .$$

We define the following events: Succ: $[v = w = \mathsf{yes}]$, and Succ': $[v' = w' = \mathsf{yes}]$. By using the Jensen inequality, we have

$$\begin{aligned} \Pr[\mathsf{Succ} \cap \mathsf{Succ}'] &= \sum_{d,a} \Pr[d, a] \cdot \Pr[\mathsf{Succ} \cap \mathsf{Succ}' \mid d,a] = \sum_{d,a} \Pr[d, a] \cdot \Pr[\mathsf{Succ} \mid d,a] \cdot \Pr[\mathsf{Succ}' \mid d,a] \\ &= \sum_{d,a} \Pr[d, a] \cdot \Pr^2[\mathsf{Succ} \mid d, a] \geq \left(\sum_{d,a} \Pr[d, a] \cdot \Pr[\mathsf{Succ} \mid d, a] \right)^2 = \delta^2(k) \enspace . \end{aligned}$$

We define two adversaries: a *collision-finder* Γ for h and a *predictor* Π that predicts $h(\mathcal{D})$.

The collision finder Γ*:* simulates the experiment above and tries to find a collision for h, based on the oracle calls (to h) performed during the experiment. The collision-finding strategy of Γ concerns in the following checks:

- If $\neg(\mathsf{Succ} \cap \mathsf{Succ}')$ then halt.
- else if $L \neq L'$ or $h(X) \notin L$ or $h(X') \notin L'$, then find a collision (z_1, z_2) (by using Lemma 2) and output (z_1, z_2) .

The predictor Π simulates: $(d,a) \leftarrow \mathsf{A}_1(1^k), X \leftarrow \mathcal{D}, (c,L) \leftarrow \mathsf{A}_2(X,a)$ and returns L.

We define the following events that are related to the experiment above:

$$\begin{aligned} \mathsf{Coll} &= \mathsf{Succ} \cap \mathsf{Succ}' \cap \left[[L \neq L'] \cup [h(X) \notin L] \cup [h(X') \notin L']\right] \\ \mathsf{Pred} &= \mathsf{Succ} \cap \mathsf{Succ}' \cap [L = L'] \cap [h(X) \in L] \cap [h(X') \in L'] \ . \end{aligned}$$

Hence, $\mathsf{Coll} \cup \mathsf{Pred} = \mathsf{Succ} \cap \mathsf{Succ}'$. The success probability $\pi(k)$ of Π is

$$\begin{aligned} \pi(k) &= \mathsf{Pr}[L \leftarrow \Pi(1^k), X' \leftarrow \mathcal{D}\colon\ h(X') \in L] \\ &= \mathsf{Pr}[(d,a) \leftarrow \mathsf{A}_1(1^k), X \leftarrow \mathcal{D}, (c,L) \leftarrow \mathsf{A}_2(X,a), X' \leftarrow \mathcal{D}, (c',L') \leftarrow \mathsf{A}_2(X',a)\colon h(X') \in L] \\ &\geq \mathsf{Pr}[\mathsf{Pred}] \ . \end{aligned}$$

Hence, if the success of Γ is denoted by $\gamma(k)$, we have

$$\gamma(k) + \pi(k) = \mathsf{Pr}[\mathsf{Coll}] + \mathsf{Pr}[\mathsf{Pred}] \geq \mathsf{Pr}[\mathsf{Coll} \cup \mathsf{Pred}] = \mathsf{Pr}[\mathsf{Succ} \cap \mathsf{Succ}'] = \delta^2(k). \qquad \square$$

E Proof of Theorem 3

Obviously, $(s_{\mathrm{R}}^0, s_{\mathrm{I}}^0) \in \beta$. Let $s_{\mathrm{R}} = (\mathfrak{L}, \mathfrak{D}, \mathfrak{C}, \mathfrak{r})$, $s_{\mathrm{I}} = (\mathfrak{L}_{\mathrm{I}}, \mathfrak{D}_{\mathrm{I}}, \mathfrak{C}_{\mathrm{I}}, \mathfrak{r}_{\mathrm{I}})$ and $(s_{\mathrm{R}}, s_{\mathrm{I}}) \in \beta$. Let $i \in I$ be an input. To prove the statement, we assume $\delta_{\mathrm{R}}(i, s_{\mathrm{R}}) \notin F_{\mathrm{R}}$, $\delta_{\mathrm{I}}(i, s_{\mathrm{I}}) \notin F_{\mathrm{I}}$, and show that $(\delta_{\mathrm{R}}(i, s_{\mathrm{R}}), \delta_{\mathrm{I}}(i, s_{\mathrm{I}})) \in \beta$ and $\lambda_{\mathrm{R}}(i, s_{\mathrm{R}}) = \lambda_{\mathrm{I}}(i, s_{\mathrm{I}})$. We have to check only two input events – $\mathsf{in}_{\mathsf{aud}} \rightarrow L$ and $\mathsf{in}_{\mathsf{cert}} \rightarrow (x, c, \tau)$, because the other inputs do not change the state of the machines and produce the same output.

If $\mathsf{in}_{\mathsf{aud}} \rightarrow (L, d)$ and $d = \mathsf{Publish}(L)$ then: [5]

- M_{R} (i.e. Repository) computes $\mathfrak{D} := \mathfrak{D} \| d$ and $\mathfrak{L} := \mathfrak{L} \| L$. No output is produced.
- M_{I} (i.e. the simulator S) computes $\mathfrak{D}_{\mathrm{I}} := \mathfrak{D}_{\mathrm{I}} \| d$, and outputs $L \overset{\triangleright}{\rightarrow} \mathsf{out}_{\mathsf{aud}}$. The ideal scheme TS_{I} (on input $\mathsf{in}_{\mathsf{aud}} \rightarrow L$) computes $\mathfrak{L}_{\mathrm{I}} := \mathfrak{L}_{\mathrm{I}} \| L$. No output is produced.

If $\mathsf{in}_{\mathsf{cert}} \rightarrow (x, c, \tau)$ and $\tau \geq |\mathfrak{D}|$ then:

- M_{R} (i.e. Verifier) sets $\mathfrak{r} := (x, c, \tau)$ and outputs $\tau \overset{\triangleright}{\rightarrow} \mathsf{out}_{\mathsf{num}}$. As a result, Repository obtains an input $\mathsf{in}_{\mathsf{num}} \rightarrow \tau$ and outputs $\mathsf{NIL} \rightarrow \mathsf{out}_{\mathsf{dig}}$ (because of $\tau \geq |\mathfrak{D}|$) and returns the control to Verifier. Having an input $\mathsf{in}_{\mathsf{dig}} \rightarrow \mathsf{NIL}$, Verifier outputs $(x, \mathsf{no}, \tau) \rightarrow \mathsf{out}_{\mathsf{res}}$.
- M_{I} (the simulator S) sets $\mathfrak{r}_{\mathrm{I}} := (x, c, \tau)$, computes $\bar{b} := \tau < |\mathfrak{D}_{\mathrm{I}}|\ \&\mathsf{Verify}(x, c, \mathfrak{D}_{\mathrm{I}}[\tau]) = \mathsf{no}$ (as $|\mathfrak{D}_{\mathrm{I}}| = |\mathfrak{D}| \leq \tau$) and outputs $(x, \mathsf{no}, \tau) \overset{\triangleright}{\rightarrow} \mathsf{out}_{\mathsf{res}}$ giving control to TS_{I}. The ideal scheme (on input $\mathsf{in}_{\mathsf{res}} \rightarrow (x, \mathsf{no}, \tau)$) computes $b := \mathsf{no}\,\&\mathsf{True}(x \in \mathfrak{L}_{\mathrm{I}}[\tau]) = \mathsf{no}$ and outputs $(x, \mathsf{no}, \tau) \rightarrow \mathsf{out}_{\mathsf{res}}$.

If $\mathsf{in}_{\mathsf{cert}} \rightarrow (x, c, \tau)$, $\tau < |\mathfrak{D}|$, and $\mathsf{Verify}(x, c, \mathsf{Publish}(\mathfrak{L}[\tau])) = \mathsf{no}$ then:

[5] Both machines do nothing if $d \neq \mathsf{Publish}(L)$.

- M_R (i.e. Verifier) sets $\mathfrak{r} := (x, c, \tau)$ and outputs $\tau \overset{\triangleright}{\to} \mathsf{out}_\mathsf{num}$. As a result, Repository obtains an input $\mathsf{in}_\mathsf{num} \to \tau$, computes $d_\tau := \mathfrak{D}[\tau]$ and outputs $d_\tau \to \mathsf{out}_\mathsf{dig}$ (returning the control to Verifier). The Verifier, having an input $\mathsf{in}_\mathsf{dig} \to d_\tau$, computes
$$b := \mathsf{Verify}(x, c, \mathfrak{D}[\tau]) = \mathsf{Verify}(x, c, \mathsf{Publish}(\mathfrak{L}[\tau])) = \mathsf{no},$$
and outputs $(x, \mathsf{no}, \tau) \to \mathsf{out}_\mathsf{res}$.
- M_I (i.e. the simulator S) sets $\mathfrak{r}_\mathrm{I} := (x, c, \tau)$ and computes (considering $\mathfrak{D}_\mathrm{I} = \mathfrak{D}$)
$$\bar{b} := \mathsf{True}[\tau <|\mathfrak{D}_\mathrm{I}|] \,\&\, \mathsf{Verify}(x, c, \mathfrak{D}_\mathrm{I}[\tau]) = \mathsf{Verify}(x, c, \mathfrak{D}[\tau]) = \mathsf{Verify}(x, c, \mathsf{Publish}(\mathfrak{L}[\tau])) = \mathsf{no},$$
and outputs $(x, \mathsf{no}, \tau) \overset{\triangleright}{\to} \mathsf{out}_\mathsf{res}$ (giving control to TS_I). The ideal scheme TS_I, on input $\mathsf{in}_\mathsf{res} \to (x, \mathsf{no}, \tau)$ computes $b := \mathsf{no} \,\&\mathsf{True}(x \in \mathfrak{L}_\mathrm{I}[\tau]) = \mathsf{no}$, and outputs $(x, \mathsf{no}, \tau) \to \mathsf{out}_\mathsf{res}$.

If $\mathsf{in}_\mathsf{cert} \to (x, c, \tau)$, $\tau <|\mathfrak{D}|$, $\mathsf{Verify}(x, c, \mathsf{Publish}(\mathfrak{L}[\tau])) = \mathsf{yes}$, and $x \in \mathfrak{L}_\mathrm{I}[\tau]$ then:

- M_R (i.e. Verifier) sets $\mathfrak{r} := (x, c, \tau)$ and outputs $\tau \overset{\triangleright}{\to} \mathsf{out}_\mathsf{num}$. As a result, Repository obtains an input $\mathsf{in}_\mathsf{num} \to \tau$, computes $d_\tau := \mathfrak{D}[\tau]$ and outputs $d_\tau \to \mathsf{out}_\mathsf{dig}$ (returning the control to Verifier). The Verifier, having an input $\mathsf{in}_\mathsf{dig} \to d_\tau$, computes [6]
$$b := \mathsf{Verify}(x, c, d_\tau) = \mathsf{Verify}(x, c, \mathfrak{D}[\tau]) = \mathsf{Verify}(x, c, \mathsf{Publish}(\mathfrak{L}[\tau])) = \mathsf{yes},$$
and $\mathfrak{C}[\tau] := \mathfrak{C}[\tau] \cup \{x\}$ and outputs $(x, \mathsf{yes}, \tau) \to \mathsf{out}_\mathsf{res}$.
- M_I (i.e. the simulator S) sets $\mathfrak{r}_\mathrm{I} := (x, c, \tau)$ and computes (considering $\mathfrak{D}_\mathrm{I} = \mathfrak{D}$)
$$\bar{b} := \mathsf{True}[\tau <|\mathfrak{D}_\mathrm{I}|] \,\&\, \mathsf{Verify}(x, c, \mathfrak{D}_\mathrm{I}[\tau]) = \mathsf{Verify}(x, c, \mathfrak{D}[\tau]) = \mathsf{Verify}(x, c, \mathsf{Publish}(\mathfrak{L}[\tau])) = \mathsf{yes},$$
$\mathfrak{C}_\mathrm{I}[\tau] := \mathfrak{C}_\mathrm{I}[\tau] \cup \{x\}$ and outputs $(x, \mathsf{yes}, \tau) \overset{\triangleright}{\to} \mathsf{out}_\mathsf{res}$ (giving control to TS_I). The ideal scheme TS_I, on input $\mathsf{in}_\mathsf{res} \to (x, \mathsf{yes}, \tau)$ computes $b := \mathsf{yes} \,\&\mathsf{True}(x \in \mathfrak{L}_\mathrm{I}[\tau]) = \mathsf{yes}$, and outputs $(x, \mathsf{yes}, \tau) \to \mathsf{out}_\mathsf{res}$.

If $\mathsf{in}_\mathsf{cert} \to (x, c, \tau)$, $\tau <|\mathfrak{D}|$, $\mathsf{Verify}(x, c, \mathsf{Publish}(\mathfrak{L}[\tau])) = \mathsf{yes}$, and $x \notin \mathfrak{L}_\mathrm{I}[\tau]$ then:

- M_R (i.e. Verifier) sets $\mathfrak{r} := (x, c, \tau)$ and outputs $\tau \overset{\triangleright}{\to} \mathsf{out}_\mathsf{num}$. As a result, Repository obtains an input $\mathsf{in}_\mathsf{num} \to \tau$, computes $d_\tau := \mathfrak{D}[\tau]$ and outputs $d_\tau \to \mathsf{out}_\mathsf{dig}$ (returning the control to Verifier). The Verifier, having an input $\mathsf{in}_\mathsf{dig} \to d_\tau$, computes
$$b := \mathsf{Verify}(x, c, d_\tau) = \mathsf{Verify}(x, c, \mathfrak{D}[\tau]) = \mathsf{Verify}(x, c, \mathsf{Publish}(\mathfrak{L}[\tau])) = \mathsf{yes},$$
and $\mathfrak{C}[\tau] := \mathfrak{C}[\tau] \cup \{x\}$ and outputs $(x, \mathsf{yes}, \tau) \to \mathsf{out}_\mathsf{res}$. The resulting state is faulty.
- M_I (i.e. the simulator S) sets $\mathfrak{r}_\mathrm{I} := (x, c, \tau)$ and computes (considering $\mathfrak{D}_\mathrm{I} = \mathfrak{D}$)
$$\bar{b} := \mathsf{True}[\tau <|\mathfrak{D}_\mathrm{I}|] \,\&\, \mathsf{Verify}(x, c, \mathfrak{D}_\mathrm{I}[\tau]) = \mathsf{Verify}(x, c, \mathfrak{D}[\tau]) = \mathsf{Verify}(x, c, \mathsf{Publish}(\mathfrak{L}[\tau])) = \mathsf{yes},$$
$\mathfrak{C}_\mathrm{I}[\tau] := \mathfrak{C}_\mathrm{I}[\tau] \cup \{x\}$ and outputs $(x, \mathsf{yes}, \tau) \overset{\triangleright}{\to} \mathsf{out}_\mathsf{res}$ (giving control to TS_I). The ideal scheme TS_I, on input $\mathsf{in}_\mathsf{res} \to (x, \mathsf{yes}, \tau)$ computes $b := \mathsf{yes} \,\&\mathsf{True}(x \in \mathfrak{L}_\mathrm{I}[\tau]) = \mathsf{yes}$, and outputs $(x, \mathsf{yes}, \tau) \to \mathsf{out}_\mathsf{res}$. The resulting state is faulty.

To conclude, in every input event and in every branch we observed, the machines produce the same output and their internal states are either both faulty or satisfy the relation β. □

[6] It is easy to check that $\mathfrak{D}[\tau] = \mathsf{Publish}(\mathfrak{L}[\tau])$ is an invariant of M_R, as well as $\mathfrak{D}_\mathrm{I}[\tau] = \mathsf{Publish}(\mathfrak{L}_\mathrm{I}[\tau])$ is an invariant of M_I.