Improving the availability of time-stamping services

Arne Ansper[1], Ahto Buldas[012], Märt Saarepera[3], and Jan Willemson[012]

[1] Cybernetica; Akadeemia 21, Tallinn, Estonia; Lai 36, Tartu, Estonia, {arne,ahtbu,jan}@cyber.ee
[2] Tartu University Department of Mathematics; Liivi 2, Tartu, Estonia
[3] Tokyo University, RCAST, marsa@hal.rcast.u-tokyo.ac.jp

Abstract. We discuss the availability questions that arise when digital time stamps are used for preserving the evidentiary value of electronic documents. We analyze the time-stamping protocols known to date and point out some weaknesses that have not been addressed so far in scientific literature. Without addressing and solving them, any advantage of the linkage-based protocols over the hash-and-sign time-stamping would be questionable. We present several new techniques and protocols for improving the availability of both the hash-and-sign and the linkage-based time-stamping services. We introduce fault-tolerant linking as a new concept to neutralize fault-sensitivity as the main weakness of linkage-based time-stamping.

1 Introduction

Time stamp is an attestation that a digital document was created at a certain time. Time stamps are essential tools for relying parties to preserve the evidentiary value of electronic data (particularly, digital signatures). Due to their responsible mission, Time-Stamping Authorities (TSAs) must be reliable – trustworthy, and available when needed. Availability threats may be as harmful as potential attacks by network hackers or dishonest behavior of other parties (repudiation etc.). For example, if TSA's server is destroyed, a large number of time stamps may get unverifiable, and therefore, relying parties may suffer from considerable monetary losses because some important documents (agreements, bills etc.) lost their evidentiary value. This seems unfair from the view-point of an interested party who has no control of the procedures running in a time-stamping server. It would thereby be reasonable if no party could affect the validity of time stamps except the relying party itself.

Regardless of their importance, availability questions have almost never been discussed in scientific literature. This paper is intended to be a contribution to filling this gap. We discuss several techniques for improving the availability of time-stamping services. Particularly, we propose protocols for using multiple

[0] Supported by the Estonian SF, grant no. 4150

time-stamping servers and argue what kind of benefits such approach may offer for both the hash-and-sign and the linkage-based systems. We also discuss methods for fighting against occasional errors in TSA's database which turn out to be the most serious threats in linkage-based time stamping systems.

In Section 2, we outline the objectives of time-stamping, the general model of time-stamping, and point out the main threats to availability. In Section 3, we analyze the time-stamping systems known to date and point out their advantages and weaknesses. In Section 4, we discuss how multiple servers can be used to improve the availability of service. In Section 5, we introduce a new concept of fault-tolerant linking – a technique against fault-sensitivity of the one-way hash computations used in linking schemes.

2 Time-stamping: objectives, model, and threats

Let $[t, t']$ be a time-interval and x, y be bit-strings. There are three basic statements that time stamps should prove:

- *Freshness* (of x at t) – x was created after t.
- *Existence* (of y at t') – y was created before t'.
- *Order* (of y and x) – y was created before x.

We call the time stamps intended to prove these statements as: (1) *freshness token*, (2) *existence token* (or *stamp*), and (3) *order token*, respectively. Freshness tokens are needed to avoid replay attacks in authentication protocols. Possibly, there exist no reliable ways of proving that x was created precisely at t. Existence tokens (or stamps) are necessary for proving that a digital signature was created before the corresponding key-identity relation was revoked. In some cases we may need to prove more than one of these statements.

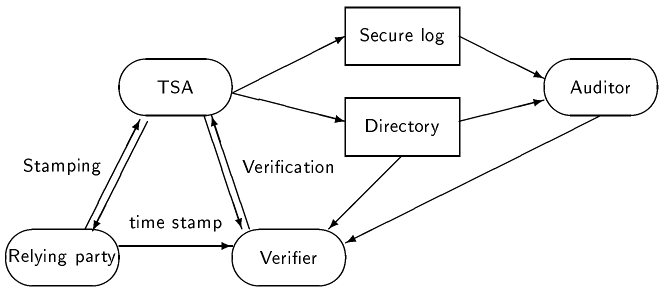

Fig. 1. General model of time-stamping.

Time-stamping is a service used by the *Relying party* to prove temporal relations to the *Verifier* (such as judge and alike). The relying party obtains time stamps from the TSA (and also takes care of them later) by using the

Stamping protocol. The Verifier uses the Verification protocol (which may require communication with the TSA) to check the correctness of time stamps presented by the relying party (Fig. 1). The TSA may also use secure logs and a public directory to enable the Auditor to audit the TSA's work. Audit reports are made available to the verifier and to the relying party. The presence of regular audit gives some additional assurance to time-stamping services.

As referred to in the Introduction, it will be fair if the evidentiary value of time stamps does not depend on third parties (other than the relying party). This is the motivation for the *compactness of evidence* principle: Relying parties possess a compact and time-proof evidence the value of which depends neither on other parties' actions nor on the events which the relying party has no sufficient control of. We discuss three such events:

[A] Broken cryptography and compromised keys. If the cryptographic mechanisms for protecting the authenticity of time stamps are compromised, there should still be a mechanism to distinguish between time stamps (1) issued by the TSA before the compromise, and those (2) created by an attacker, using compromised cryptography. Hence, all the time stamps issued so far can be called into question and cannot further be used as evidence.

[B] Service unavailability. Time-stamping service itself gets unavailable for a while. Relying parties are not able to obtain time stamps for documents they want to preserve as evidence. Such an accident may be causal, for example, in a stock market computer system where time stamps are used to arrange stockbrokers' requests. Unavailability is often caused by the denial of service attacks which are possible if the communication (or security) protocols are poorly designed.

[C] Loss of server's data. A portion of data in a time-stamping server is destroyed and a fraction of time stamps becomes unverifiable. This means that a large number of documents may lose their evidentiary value, and therefore, relying parties may suffer from considerable monetary losses. This type of unavailability may be caused by occasional errors in TSA's server. The most important reason that makes this threat more serious than the previous one is that neither the server nor the relying parties may notice that errors have occurred and the server uselessly continues its work.

In this paper, we analyze which of those threats are encountered in each time-stamping system. We also propose new techniques for overcoming these threats.

3 Time-stamping systems: overview

Preliminaries and notation. By $\mathrm{Sig}_A\{X_1, \ldots, X_m\}$ we mean a digital signature created by A on the ordered list of messages $X_1, \ldots, X_m$. Sometimes, we inherently assume that A uses a signature scheme with message recovery. By a collision-free hash function h, we mean a polynomial-time function family such that it is computationally infeasible to find two arguments $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$. Exact mathematical definition of a hash function is unnecessary for understanding the subject of this paper and is omitted.

Technical assumptions. We only deal with time-stamping systems with one or more central authorities, though there exist protocols without central authorities [1, 2]. We also assume that, in order to prevent unreasonable communication, documents are always hashed before they are included into time-stamping requests.

3.1 Absolute (hash-and-sign) time stamps

Absolute (hash-and sign) time stamps are tokens (signed by the TSA) which comprise a document (or a hash of the document) and a date/time represented as a number. Security of this scheme is based on the assumption that the TSA has a precise enough clock device and is completely trustworthy.

To obtain a freshness token $\mathcal{H}$, a client A (Alice) sends a request to the TSA. The TSA signs the current time t and sends $\mathcal{H} = \mathrm{Sig}_{\mathrm{TSA}}\{t\}$ back to A. For example, given a message $\sigma = \mathrm{Sig}_A\{X, \mathcal{H}\}$, Bob is able to verify that X was signed by Alice after t. Note that even if the TSA is trusted and trustworthy, the freshness token does not prove that σ was created precisely at t.

To obtain a stamp for a bit-string x (for example, $x = \sigma$), a client B (Bob) sends x to the TSA. The TSA adds the current time and date t' to x and sends $\mathcal{T} = \mathrm{Sig}_{\mathrm{TSA}}\{x, t\}$ back to B. The triple $(\mathcal{H}, \sigma, \mathcal{T})$ proves that A signed X during the interval $[t, t']$.

Main concern: key compromise. In systems with a single TSA there seems to be no efficient solution to this problem. The best solution seems to be that the TSA stores all time stamps it ever issues. If then the key is compromised, the TSA signs all time stamps with a new key. This is impractical because of high storage and computational complexity. Tamper-proof hardware may be used to prevent the key compromise. The hardware module may even generate a key-pair for the TSA and never let the private component outside the module. However, this is neither a completely usable solution because the signature scheme itself may be broken and the key length insufficient. In Surety's white paper [8] they even conclude that using keyed cryptography for time-stamping is extremely flawed and only the keyless cryptography can do the job. However, we do not completely agree with this categorical statement. We will show in this paper that the key change is practical in a multi-server case.

3.2 Auditable relative time-stamping

There is a substantive relationship between absolute time stamps and trust. People have unconsciously accepted the concept of time as a number and they hardly realize that actually the relation between physical time and numbers is almost as artificial as the relation between people and their public keys (which are numbers as well!). Thereby, the relationship between time and numbers cannot be fixed reliably without using trusted third parties. However, this does not mean there is no temporal measure which is independent of trusted third parties! Indeed, under some assumptions about computational intractability (one-wayness of hash functions), we have relative temporal measure that uses no trust assumptions.

Let h be a collision resistant one-way hash function and y be a bit-string which was published in a newspaper on February 20, 2001. If we are sent a bit-string x satisfying the relation $y = h(x)$ then we are convinced (because of one-wayness) that x was known to somebody (or stored into a computer) before y was created. Therefore, we also know for sure that x was created before February 20, 2001. More generally, if

$$y = \ell(x, x_1, \ldots, x_n), \tag{1}$$

where ℓ is an arbitrary hash formula (e.g., $h(h(x, x_1), h(x_2, x_3)))$, then $(x_1, \ldots, x_n)$ is a proof that x was created before y. Let $x = \mathrm{Sig}_A\{X\}$ be Alice's signature on X and $\sigma = \mathrm{Sig}_B\{Y, y\}$ be Bob's signature on Y which also comprises a bit-string y, such that equation (1) holds. Then $(x_1, \ldots, x_n)$ is an undeniable proof that Alice signed X before Bob signed Y. Note that the proof itself uses only keyless cryptography and its validity is thereby not affected by the key compromise. This is the main idea of linking first proposed by Haber et al [7].

The TSA maintains a secure log file $(\ell_0, \ell_1, \ldots, \ell_n, \ldots)$ created by using a collision-free hash function h with k-bit output. After each request x_n the TSA computes a new value of ℓ_n using the following recursive formula:

$$\ell_n = h(x_n, \ell_{n-1}) \tag{2}$$

The most important property of the linking scheme is that the value of each log item ℓ_n depends in one-way manner on all the previous items $\ell_0, \ldots, \ell_{n-1}$. If ℓ_n was published in a newspaper on February 20, 2001, then: (1) the previous values cannot be modified without the possibility of detection by an Auditor; and (2) $\ell_0, ..., \ell_n$ may be used as existence tokens for $x_0, \ldots, x_n$, respectively.

To obtain a freshness token, Alice sends a request to the TSA. The TSA sends back the most recent ℓ_m. To obtain a stamp for a bit-string x_n, an interested party B (Bob) sends x_n to the TSA. The TSA computes a new value for ℓ_n using (2). Finally, the TSA sends ℓ_{n-1} back to Bob in order to make him able to compute ℓ_n from x_n, and optionally, uses a short-term signature key to authenticate ℓ_n. So, an existence token is obtained through the following protocol:

$$\boxed{\begin{array}{ll} 1.\ B \to \text{TSA:} & x_n \\ 2\ \ \text{TSA computes:} & \ell_n = h(x_n, \ell_{n-1}) \\ 3.\ \text{TSA} \to B\text{:} & \ell_{n-1},\ [\,\mathrm{Sig}_{\mathrm{TSA}}\{\ell_n\}\,]. \end{array}} \tag{3}$$

Here and further, the square brackets mean that the signature is optional in this protocol. From time to time (say, weekly), the TSA publishes the most recent ℓ_n in the Directory (Fig. 1). After that, the TSA and no-one else is able to modify the chain $\ell_0, \ldots, \ell_{N-1}$ of previous log items. The secure log may also be made widely public. For verifying the order of ℓ_m and ℓ_n $(m < n)$, the Verifier obtains a list $\mathcal{T}_{m,n} = (x_{m+1}, x_{m+2}, \ldots, x_n)$ and performs $n - m$ hash-steps (Fig. 2). The list $\mathcal{T}_{m,n}$ is an undeniable proof that ℓ_m was issued before ℓ_n.

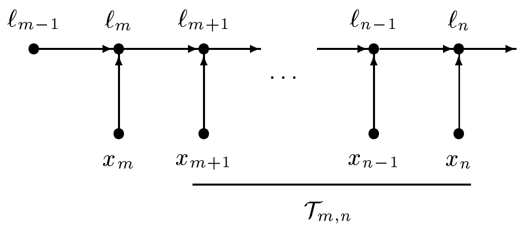

Fig. 2. Linear linking scheme.

Remark: What do signatures add to the linked service? At first sight, the TSA's signature on ℓ_n in Protocol (3) seems redundant since the signature is unnecessary for the comparison of time stamps. However, the TSA's signature is still valuable from the view-point of availability as a method of authenticating the TSA by clients. A malicious party may act as the TSA and thereby, the interested party (Bob) cannot be sure about the validity of the time stamp. Another reason to use signatures is that a malicious TSA may manipulate with the temporal order before it publishes a log item in the Directory. As the value of ℓ_n comprises the whole previous history, the signature of the TSA on ℓ_n can be taken as a temporary commitment. Any modifications in the list $(\ell_0, \ldots, \ell_{n-1})$ can be detected and proved by Bob, even if no ℓ_N $(N \geq n)$ has been published yet.

Main drawback: verification cost. Reliable (trust-free) verification of temporal relations may require a large amount of computation and storage, because both are linear functions in $\mid n - m \mid$. The Verifier should download a large amount of data for each verification which may cause huge traffic on the Internet. If a trusted server is used to perform the computation, we have almost the same trust problems as in the absolute time-stamping case. We have very much the same "trust versus communication" problem here as in the public key infrastructure – revocation lists (CRL) require communication, whereas on-line status protocol (OCSP) requires trust. Regular audit may, to some extent, increase reliability of the service. However, an assumption about a trusted Auditor is very much the same as an assumption about a trusted TSA.

The compactness of evidence principle is, thereby, almost unachievable, because it would require a huge amount of storage in the relying party's side. For the service being practical, the Verifier needs communication with the TSA and hence the evidentiary value of time stamps depends on the availability of TSA's service.

3.3 Time certificates

Time certificates approach, which was first proposed by Pinto and Freitas [10] and independently by Buldas, Laud, Lipmaa and Villemson [4] tries to reduce the communication by saving a part of the linkage data as a meta-part $\tau(X)$ (called *time certificate*) of the time-stamped document X. The purpose of a time certificate $\tau(X)$ is to fix the temporal position of X in a reliable way, so that $(\tau(X), \tau(Y))$ is always a compact piece of evidence for temporal comparison of X and Y (Fig. 3).

We now briefly describe how to use linear linking scheme to create time certificates. Let ℓ_N be the most recent log item which is published, and $x_0, x_1, \ldots, x_N$ be the bit-strings time-stamped so far. It is easy to see that we may use a pair $\tau(x_n) = (\ell_{n-1}, \mathcal{T}_{n,N})$ as a time certificate for x_n. Indeed, let $m < n$ and $\tau(x_m) = (\ell_{m-1}, \mathcal{T}_{m,N})$ be another certificate. Then, by the definition,

$$\mathcal{T}_{m,n} = (x_{m+1}, \ldots, x_n) \subseteq (x_{m+1}, \ldots, x_N) = \mathcal{T}_{m,N}$$

which means that the pair $(\tau(x_m), \tau(x_n))$ indeed comprises a proof that x_m was time-stamped before x_n.

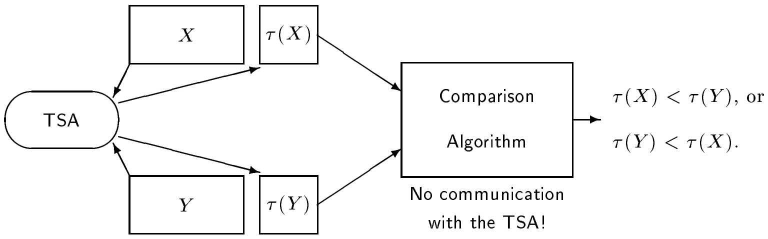

Fig. 3. Off-line comparison of time certificates.

As we mentioned, the linear linking scheme (2) is not practical for using time certificates because of certificates' size (linear in the total number of time stamps). More complex linking schemes presented by Buldas et al [4, 3, 5, 6] make certificates practical because their size in those schemes is logarithmic in the total number of time stamps.

In general, the protocols are almost the same as in the linear scheme, except that after each request x_n the TSA computes two new values: (1) ℓ_n is a bit-string the length of which is equal to the output length k of h; (2) $\mathcal{H}_n$ is a sequence of bit-strings of length k. Computations use the following recursive formulae:

$$\ell_n = \mathcal{L}(x_n, \mathcal{H}_{n-1}), \qquad \mathcal{H}_n = \mathcal{H}(\ell_n, \mathcal{H}_{n-1}), \tag{4}$$

where $\mathcal{H}$ and $\mathcal{L}$ are polynomial-time algorithms consisting of one-way hash computations. Note that by taking $\mathcal{H}_n = \{\ell_n\}$ and $\ell_n = h(x_n, \ell_{n-1})$ we get the linear scheme. The freshness token in the general scheme is $\mathcal{H}_n$. Existence token can be obtained through the following protocol:

$$\boxed{\begin{array}{ll} 1.\ B \to \text{TSA:} & x_n \\ 2\ \ \text{TSA computes:} & \ell_n = \mathcal{L}(x_n, \mathcal{H}_{n-1}), \quad \mathcal{H}_n = \mathcal{H}(\ell_n, \mathcal{H}_{n-1}) \\ 3.\ \text{TSA} \to B\text{:} & \mathcal{H}_{n-1},\ [\,\mathrm{Sig}_{\mathrm{TSA}}\{\ell_n\}\,]. \end{array}} \tag{5}$$

Time certificate for x_n is a pair $\tau(x_n) = (\mathcal{H}_{n-1}, \mathcal{T}_{n,N})$, where $\mathcal{T}_{n,N}$ is a set of hash values, comprising a proof that the value of ℓ_N depends on ℓ_n. Linking algorithms $\mathcal{L}$ and $\mathcal{H}$ can be chosen so that for any $m < n$ the certificates $\tau(x_m) = (\mathcal{H}_{m-1}, \mathcal{T}_{m,N})$ and $\tau(x_n) = (\mathcal{H}_{n-1}, \mathcal{T}_{n,N})$ together are enough to construct a verifiable hash-chain from ℓ_m to ℓ_n. At the same time, the size of a certificate is logarithmic in N. One such linking scheme – *threaded tree* [5] – is described in Appendix A.

One important property of time certificates is that they can be extended, i.e. for any $N_1 > N$ the Relying party (say Bob) may request the TSA for a proof $\mathcal{T}_{N,N_1}$ and then extend the certificate $\tau(x_n)$ from $\tau(x_n) = (\mathcal{H}_{n-1}, \mathcal{T}_{n,N})$ to $(\mathcal{H}_{n-1}, \mathcal{T}_{n,N_1})$. For extension, Bob sends TSA a request which contains two numbers (N, N_1). The TSA answers with $\mathcal{T}_{N,N_1}$ which comprises a one-way link from ℓ_N to ℓ_{N_1}. The answer may also be completed with the TSA's signature $\mathrm{Sig}_{\mathrm{TSA}}\{\ell_{N_1}\}$. Therefore, the extension goes as follows:

$$\boxed{\begin{array}{l} 1.\ B \to \text{TSA: } (N, N_1) \\ 2.\ \text{TSA} \to B\text{: } \mathcal{T}_{N,N_1},\ [\,\mathrm{Sig}_{\mathrm{TSA}}\{\ell_{N_1}\}\,] \end{array}} \tag{6}$$

We mention without giving a proof that in the threaded tree linking scheme [5] described in Appendix A, we can define an easily-computable composition operation $\circ$, so that

$$\mathcal{T}_{n,N_1} = \mathcal{T}_{n,N} \circ \mathcal{T}_{N,N_1}.$$

The most important thing here is that Bob can extend a time stamp several times, whereas its size will stay logarithmic. This is not the case if we just concatenate $\mathcal{T}_{n,N}$ and $\mathcal{T}_{N,N'}$, because doing so for numerous times, we end up in the linear certificate size.

Therefore, if a set of time stamps are extended to the same published time stamp their temporal order can be determined without the TSA. This is an important property from the view-point of availability. For example, the TSA may require that all the time stamps older than January 1, 2000 were extended to N which was the first time stamp issued on that date. If the relying parties indeed do so, the values $\ell_0, \ldots, \ell_{N-1}$ may be deleted from the TSA's server to save storage space.

Main advantages of the time certificates approach over the previous ones are that (1) Time certificates may always be extended to the most recently published log values. Thereby, users are able to audit the TSA by themselves. (2) Time certificates can be used to verify temporal order of documents off-line, without communicating with the TSA.

Main drawback: increased fault-sensitivity. Computation of log items ℓ_n is (and should be!) extremely fault sensitive. Hence, there is always a concern that errors in the log file will propagate to the future. As a result, the hash-chain splits into two parts, whereas time stamps in different parts would be incomparable. If we compare the formulas of linear linking (2) and general linking (4), we notice that linear linking is less susceptible to error propagation because the value of ℓ_n depends only on ℓ_{n-1}, whereas in the general linking scheme, $\mathcal{H}_{n-1}$ may comprise relatively old ℓ_m, with $m \ll n$. Therefore, if something happens to ℓ_m in the interval between creation of ℓ_m and ℓ_{n-1}, computing ℓ_n in the next step leads us to erroneous log file. So far, almost no attention has been paid in scientific literature to this concern. In Section 5 of this paper we propose a method of fault-tolerant linkage which significantly reduces the danger of fault propagation.

3.4 Usage example: Time stamps for digital signatures

Time-stamping of digital signature begins with obtaining a *freshness token* from the TSA. The signer A (Alice) sends a request to the TSA and receives the most recent freshness token $\mathcal{H}_m$. In the hash-and-sign scheme, $\mathcal{H}_m = \mathrm{Sig}_{\mathrm{TSA}}\{t\}$, where t is the current time. Alice first concatenates $\mathcal{H}_m$ with the message she wants to sign and then applies the signature algorithm to the concatenation. As a result, she has a signature $\sigma = \mathrm{Sig}_A\{X, \mathcal{H}_m\}$. Suppose B (Bob) is a relying party who received σ and wants that σ would retain its provable authenticity. Usually, this means that B should be able to prove that σ was created *before* the Alice's public key was revoked. To obtain a time stamp, Bob computes a hash $x = h(\sigma)$ and sends x to the TSA as a time stamp request.

A time-stamped signature consumes both the freshness token $\mathcal{H}$ and the stamp $\mathcal{T}$ issued by the TSA. In naive time-stamping systems, the stamp $\mathcal{T}$ is computed as follows:

$$\mathcal{T} = \mathrm{Sig}_{\mathrm{TSA}}\{\mathrm{Sig}_A\{X, \mathcal{H}_t\}, t'\},$$

where $\mathcal{H}_t$ is either $\mathrm{Sig}_{\mathrm{TSA}}\{t\}$, if Alice obtained a freshness token, or t, if Alice did not use the freshness token and added time/date to the message herself. In certain applications this would be sufficient. Naturally, in this case t should be interpreted as the time/date Alice *declares* she signed X at. Note that t (as opposed to $\mathrm{Sig}_{\mathrm{TSA}}\{t\}$) under Alice's signature does not prove that the signature was actually created at t (or after).

In a linkage-based time-stamping system, the TSA sends back the most recent linking item ℓ_n. The pair $(\mathcal{H}_m, L_n)$ is called a preliminary time stamp for σ. For extension of this time stamp, Bob sends TSA the request which contains two numbers (n, N), where $N > n$. The TSA answers with $\mathcal{T}_{n,N}$ (and optionally, with the signature $\mathrm{Sig}_{\mathrm{TSA}}\{L_N\}$). The triple $(\mathcal{H}_m, \mathcal{T}_{n,N}, \mathrm{Sig}_{\mathrm{TSA}}\{L_N\})$ is called a *time certificate* for σ.

One must be careful in time-stamping digital signatures. Signature algorithms behave almost like one-way functions. However, if the key-space is also considered as part of the input space, the signature scheme as a function is not collision-free. Massias et al showed [9] that an attacker may generate a weak RSA key and back-date signatures created with that key. Note that this attack would work in both the hash-and-sign and in the linkage-based time-stamping. To prevent this attack, it is sufficient to time stamp a hash $h(X, \mathrm{Sig}_A\{X\})$ instead of time-stamping a pure signature $\sigma = \mathrm{Sig}_A\{X\}$,.

4 Time-stamping with multiple servers.

Using multiple servers is an obvious approach when fighting against service unavailability and loss of data. In this section, we discuss what kind of benefits this approach would give in the case of absolute time-stamping and in linkage-based time-stamping. It turns out that the motivations of using multiple servers are somewhat different in those cases.

As we show in this section, using multiple servers is both (1) a prevention measure and a (2) recovery measure. It helps to keep time stamping services available to clients, and in some cases, to restore the evidentiary value of time stamps if the keys are compromised (in absolute time-stamping) or if TSA's database is lost (in linkage-based time-stamping).

Assumptions. Suppose we have s mutually trusting time-stamping servers TSA_1, ..., TSA_s. We emphasize that it is not assumed that the servers are completely trusted by all clients. To obtain a time stamp, the relying party may interact with all of these servers. Servers' interaction is assumed to be invisible to clients and be protected using a standard authentication protocol (such as SSL). We assume that at every moment, there is at least one server available to clients.

4.1 Absolute time-stamping with multiple servers

It turns out to be relatively easy to improve the availability of absolute time-stamping just by putting two or three servers together. We need servers' interaction only to synchronize their clocks and in key change scenarios. Using multiple TSAs would enable us to solve the key change problem without storing all the time stamps in servers.

To obtain a freshness token, Alice (A) sends a request to all three servers. The TSA_i signs the current time t_i and sends $\mathcal{H}^i = \mathrm{Sig}_{\mathrm{TSA}_i}\{t_i\}$ back to A:

$$\boxed{\begin{array}{ll} 1.\ \forall i: & A \to \mathrm{TSA}_i\text{: request}_i \\ 2.\ \forall i: & \mathrm{TSA}_i \to A\text{: } \mathcal{H}^i = \mathrm{Sig}_{\mathrm{TSA}_i}\{t_i\} \end{array}} \qquad (7)$$

A s-tuple $\mathcal{H} = (\mathcal{H}^1, \ldots, \mathcal{H}^s)$ is a freshness token. If Alice signs a message X together with freshness token $\mathcal{H}$ then Bob can verify that the signature $\sigma = \mathrm{Sig}_A\{X, \mathcal{H}\}$ was created after $t = \max\{t_1, \ldots, t_s\}$.

To obtain a stamp for a bit-string x (for example, $x = \sigma$), Bob (B) sends x to all s servers. The TSA_i adds the current time t'_i to x, signs the result and sends it back to B:

$$\boxed{\begin{array}{ll} 1.\ \forall i: & B \to \mathrm{TSA}_i\colon x \\ 2.\ \forall i: & \mathrm{TSA}_i \to B\colon \mathcal{T}^i = \mathrm{Sig}_{\mathrm{TSA}_i}\{x, t_i\} \end{array}} \tag{8}$$

A s-tuple $\mathcal{T} = (\mathcal{T}^1, \ldots, \mathcal{T}^s)$ is the stamp of x. $\mathcal{T}$ proves that σ was created before $t' = \min\{t'_1, \ldots, t'_s\}$. Accordingly, the triple $(\mathcal{H}, \sigma, \mathcal{T})$ proves that A signed X during $[t_i, t'_i]$, assuming that the signature key of TSA_i was valid and the TSA_i itself is trustworthy.

Key change. If the signature key of TSA_1 is compromised, the components $\mathcal{H}_1$ and $\mathcal{T}_1$ signed with this key are no more reliable. The TSA_1 generates a new key and publishes it in a reliable and widely witnessed way. After that, clients may use a *renewal* protocol for replacing the components of $\mathcal{T}$ signed with the old key with components signed with the new key. The renewal protocol runs as follows:

$$\boxed{\begin{array}{ll} 1.\ B \to \mathrm{TSA}_1\colon & \mathrm{Sig}_{\mathrm{TSA}_{1,\mathsf{old}}}\{x, t_1\}, \mathrm{Sig}_{\mathrm{TSA}_2}\{x, t_2\}, \ldots \mathrm{Sig}_{\mathrm{TSA}_s}\{x, t_s\} \\ 2.\ \mathrm{TSA}_1\colon & \text{verifies signatures } \mathrm{Sig}_{\mathrm{TSA}_2}\{x, t_2\}, \ldots, \mathrm{Sig}_{\mathrm{TSA}_s}\{x, t_s\} \\ & \text{if the signatures are valid and given on the same } x\colon \\ 3.\ \mathrm{TSA}_1 \to B\colon & \mathrm{Sig}_{\mathrm{TSA}_{1,\mathsf{new}}}\{x, \min\{t_2, \ldots, t_s\}\}. \end{array}} \tag{9}$$

Therefore, TSA_1 is able to renew time stamps without storing all the previously issued time stamps.

Note also that signing old time stamps with a new key is not usable for freshness tokens added to digitally signed documents as signed attributes. To explain this, suppose that we have a signature $\sigma = \mathrm{Sig}_A\{X, \mathcal{H}\}$ with a freshness token $\mathcal{H}$ and the signature key of the TSA is compromised. We cannot just replace the key and sign $\mathcal{H}$ again with a new key because then σ would not be verifiable any more. Signatures are one-way operations and we cannot modify the content of a signed document without violating the signature. Freshness tokens signed with a compromised key are equivalent with plain time stamps created by the signer herself. As we will see later, relative time stamps do not suffer from this concern.

4.2 Linking with multiple servers

As linking-based time stamps are keyless, the key compromise is not a motivation of using multiple servers. Moreover, there are many wide-spread techniques for increasing the reliability of storage on the hardware level (e.g. RAID). Thereby, designing special protocols for time-stamping with multiple servers may seem unnecessary. Indeed, we may just use s identical copies of a time-stamping server that uses a linking scheme, such that the clients would even not know that actually there are s servers. However, if the motivation of using multiple servers is to increase the availability of service, there should be multiple processes (running in separate machines) for creating new linking items when the request from a client is received. Just holding several copies is insufficient. One should also guarantee that each request x_n from a client is transmitted to all servers identically. If anything here goes wrong, the servers end up with different linking chains. Considering the nature and purpose of time-stamping services, we cannot completely exclude the possibility of incoherence between the servers because its probability increases as we consider large time-frames. In this paper, we do not assume that each request x_n is certainly received by all servers identically. Therefore, the linking chains (log files) maintained by the servers may be different. The protocol we describe is simple and does not require any additional means of reliable storage. Therefore, we increase the reliability. However, we pay the price in the time certificate size which increases approximately s times. The protocol may be of interest, if the documents time-stamped have a relatively high monetary value. Otherwise, standard tools (RAID etc.) would do their job well.

Assumptions and notation. Each server uses a linking scheme described by general formulae (4). The n-th log item created by the i-th server TSA_i will be denoted as ℓ_n^i. Similarly, we use $\mathcal{H}_m^i$ and $\mathcal{T}_{n,N}^i$ to denote, respectively, the freshness tokens and the relative proofs generated by TSA_i.

Freshness tokens and stamps. To obtain a freshness token, Alice sends each of the servers a request and obtains answers from each of them independently. Each answer consists of s hash formulae. For example, an answer from TSA_1 is

$$(\mathcal{H}^1_{m_{11}}, \mathcal{H}^2_{m_{12}}, \ldots, \mathcal{H}^s_{m_{1s}}),$$

where $\mathcal{H}^j_{m_{1j}}$ is the freshness token issued by TSA_j which TSA_1 knows are the most recent ones. These values are distributed between the servers using protocols which we describe later. If a server TSA^i is down and does not answer, we set $m_{i1} = m_{i2} = \ldots = m_{is} = 0$. The freshness token protocol runs as follows:

$$\boxed{\begin{array}{lll} 1.\ \forall i: & A \to \mathrm{TSA}_i: & \mathrm{request}_i \\ 2.\ \forall i: & \mathrm{TSA}_i \to A: & (\mathcal{H}^1_{m_{i1}}, \mathcal{H}^2_{m_{i2}}, \ldots, \mathcal{H}^s_{m_{is}}) \\ 3.\ \forall j: & A \text{ computes: } & m_j = \max\{m_{1j}, m_{2j}, \ldots, m_{sj}\} \\ 4.\ A \text{ computes:} & & \mathcal{H} = (\mathcal{H}^1_{m_1}, \mathcal{H}^2_{m_2}, \ldots, \mathcal{H}^s_{m_s}) \end{array}} \qquad (10)$$

Existence tokens are obtained almost in the same way as in the one-server case. The Relying party sends time stamp requests to all servers and obtains time stamps from all of them except those being inaccessible at the moment.

$$\boxed{\begin{array}{ll} 1.\ \forall i: & B \to \mathrm{TSA}_i\text{: } x \\ 2.\ \forall i: & \mathrm{TSA}_i \to B\text{: } \ell^i_{n_i - 1}, [\sigma_i = \mathrm{Sig}_{\mathrm{TSA}_i}\{\ell^i_{n_i}\}] \end{array}} \qquad (11)$$

Time-certificate $\tau(x)$ for x is a s-tuple $((\mathcal{H}^1_{n_1}, [\sigma_1]), (\mathcal{H}^2_{n_2}, [\sigma_2]), \ldots, (\mathcal{H}^s_{n_s}, [\sigma_s]))$ the components of which are ordinary time stamps (already familiar to us from the one-server case).

Extension of a time-certificate is performed on a component-wise basis. Each component is extended using the same protocol as introduced in the one-server case. For extension of the i-th component of a time-certificate, the relying party (Bob) sends TSA_i the request which contains two numbers (n_i, N_i) with $N_i > n_i$. The TSA_i answers with $\mathcal{T}^i_{n_i,N_i}$ and, optionally, with the signature $\mathrm{Sig}_{\mathrm{TSA}_i}\{\ell^i_{N_i}\}$.

$$\boxed{\begin{array}{l} 1.\ B \to \mathrm{TSA}_i\colon (n_i, N_i) \\ 2.\ \mathrm{TSA}_i \to B\colon \mathcal{T}_{n_i,N_i},\ [\mathrm{Sig}_{\mathrm{TSA}_i}\{\ell^i_{N_i}\}] \end{array}} \tag{12}$$

Boot. If TSA_1 wakes up, it checks whether it has stored values of $\mathcal{H}^1_{n_{11}}$, $\mathcal{H}^2_{n_{12}}$, ... , $\mathcal{H}^s_{n_{1s}}$. These values may be missing either because the server has never been up before or because the linkage data has been destroyed in the last crash. In that case, TSA_1 sets $n_{11} = n_{12} = \ldots = n_{1s} = 0$. After that, the TSA_1 obtains (using protocol (10)) freshness tokens form other servers and starts linking from the latest value of $\ell_{n_{s1}}$ the other servers know.

$$\boxed{\begin{array}{lll} 1.\ \forall j > 1\colon & \mathrm{TSA}_1 \to \mathrm{TSA}_j\colon & \text{request} \\ 2.\ \forall j > 1\colon & \mathrm{TSA}_j \to \mathrm{TSA}_1\colon & (\mathcal{H}^1_{n_{j1}}, \mathcal{H}^2_{n_{j2}}, \ldots, \mathcal{H}^s_{n_{js}}) \\ 3.\ \forall j\colon & \multicolumn{2}{l}{\mathrm{TSA}_1 \text{ computes: } n_{1j} := \max\{n_{1j}, n_{2j}, \ldots, n_{sj}\}} \end{array}} \tag{13}$$

Let $n = n_{11}$. If the database was indeed destroyed during the last crash, TSA_1 sets $x_{n+1} = h(\mathcal{H}^2_{12}, \ldots, \mathcal{H}^s_{1s})$ and creates ℓ^1_{n+1} and $\mathcal{H}^1_{n+1}$ using formulae (4). If time-certificates older than ℓ^1_n are then completed with the list $(\mathcal{H}^2_{12}, \ldots, \mathcal{H}^s_{1s})$, then we are again able to compare (off-line) time stamps issued by TSA_1 with older time stamps issued by other servers before the crash.

New linking item. If the server TSA_i creates a new linking item $L_{n_{ii}+1}$, it computes $\mathcal{H}^i_{n_{ii}+1}$ and sends it immediately to other servers. For example, if $i = 1$ the following protocol is completed:

$$\boxed{\begin{array}{lll} 1.\ \mathrm{TSA}_1 \text{ computes:} & & n_{11} := n_{11} + 1 \\ 2.\ \forall j > 1\colon & \mathrm{TSA}_1 \to \mathrm{TSA}_j\colon & \mathcal{H}^1_{n_{11}} \\ 3.\ \forall j > 1\colon & \mathrm{TSA}_j \text{ computes:} & \mathcal{H}^1_{n_{j1}} := \mathcal{H}^1_{n_{11}} \end{array}} \tag{14}$$

5 Fault tolerant linking

As mentioned above, the linking items ℓ_n may get corrupted, either because of occasional errors in hardware or bugs in programs running in the same computer with the TSA software. In linking schemes such errors would propagate to the future and affect the correctness of a large number of time stamps. This threat is even more dangerous than a complete destruction of the TSA's database because the TSA is unaware of the disaster and the service may stay unavailable for a long time. Therefore, some detection measures would be desirable. Error detection codes are the most widespread means against the loss/corruption of data. However, instead of including additional data fields into the linking scheme, we may use the linking scheme itself to detect and correct occasional losses of data. The crucial idea is that a collision-resistant hash function itself is a very efficient error detection code. According to equations (4), we define error detection codes for ℓ_n and for $\mathcal{H}_n$ as follows:

$$\mathsf{Code}(\ell_n) = (x_n, \mathcal{H}_{n-1}), \qquad \mathsf{Code}(\mathcal{H}_n) = (\ell_n, \mathcal{H}_{n-1}). \tag{15}$$

Verifying the code just means verifying the equations (4). Before computing the values of ℓ_n and $\mathcal{H}_n$ by formulae (4), the TSA checks the code $\mathsf{Code}(\mathcal{H}_{n-1})$. If the code is not OK (i.e. if $\mathcal{H}_{n-1} \neq \mathcal{H}(\ell_{n-1}, \mathcal{H}_{n-2})$), then the TSA concludes that the database is corrupted.

Note that the codes will work only if the errors are occasional, i.e. are not caused intentionally by an attacker. For example, if an attacker modifies ℓ_{n-1} and $\mathcal{H}_{n-2}$ and computes a new value for $\mathcal{H}_{n-1}$ using (4), then the error detection procedure we described would not detect any changes because $\mathsf{Code}(\mathcal{H}_{n-1})$ is correct. Therefore if errors are occasional, any error which affects future computations is detected at the very next linkage step. Indeed, equations (4) show that if the value of $\mathcal{H}_{n-1}$ is correct and the request x_n is obtained correctly, then the values ℓ_n, $\mathcal{H}_n$ can also be computed correctly. Any error which does not change $\mathcal{H}_{n-1}$ has no influence on future computations.

Note also that, in principle, the TSA may also try to correct errors by using the codes recursively. This idea needs further research and is not discussed in this paper.

6 Conclusions

This paper indicates that practical problems related to the reliability of digital time-stamping services are far from being completely solved. One of such problems is availability which has not been sufficiently addressed in scientific literature so far. We discussed the availability concerns of both the hash-and-sign and the linkage-based time-stamping systems. We showed how the use of multiple servers eliminates one of the most important threats in hash-and-sign time-stamping – the TSA key compromise. We pointed out a new weakness in linkage-based time-stamping – fault-sensitivity – which arises in its full strength in binary linking schemes [4, 3, 5]. To overcome the fault-sensitivity concern, we proposed a new approach – fault-tolerant linking – which in our opinion deserves future research.

Acknowledgements

The authors are grateful to Helger Lipmaa and to anonymous referees whose comments helped to improve the final version of this paper.

References

1. Josh Benaloh and Michael de Mare. Efficient broadcast time-stamping. Technical Report 1, Clarkson University Department of Mathematics and Computer Science, August 1991.

2. Josh Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Advances in Cryptology – Eurocrypt'93*, volume 765 of *LNCS*, pages 274–285. Springer-Verlag, 1994.
3. Ahto Buldas and Peeter Laud. New linking schemes for digital time-stamping. In *Proc. 1st International Conference on Information Security and Cryptology – ICISC'98*, pages 3–13, Seoul, Korea, December 1998.
4. Ahto Buldas, Peeter Laud, Helger Lipmaa, and Jan Villemson. Time-stamping with binary linking schemes. In *Advances in Cryptology – CRYPTO'98*, volume 1462 of *LNCS*, pages 486–501, Santa Barbara, 1998. Springer-Verlag.
5. Ahto Buldas, Helger Lipmaa, and Berry Schoenmakers. Optimally efficient accountable time-stamping. In *Public Key Cryptography – PKC'2000*, volume 1751 of *LNCS*, pages 293–305, Melbourne, Australia, January 2000. Springer-Verlag.
6. Helger Lipmaa. Secure and Efficient Time-Stamping Systems. PhD thesis. *Dissertationes Mathematicae Universitatis Tartuensis.* Tartu University Press, 1999.
7. Stuart Haber and W.Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991.
8. *Secure Time/Date Stamping in a Public Key Infrastructure*. Available at `http://www.surety.com/index-nn.html`
9. Henri Massias, Xavier Serret, and Jean-Jaques Quisquater. Timestamps: Main issues on their use and implementation. In *Proceedings of IEEE 8th International Workshops on enabling Technologies: Infrastructure for Collaborative Enterprises - Fourth International Workshop on Enterprise Security*, pages 178–183, June 1999. ISBN 0-7695-0365-9.
10. Fernando Pinto and Vasco Freitas. Digital time-stamping to support non repudiation in electronic communications. In *Proc. SECURICOM'96 – 14th worldwide Congress on Computer and Communications Security and Protection*, CNIT, pages 397–406, Paris, June 1996.

7 Appendix A: Linking scheme

For the readers interested in details, we present as an example the linking scheme construction given in [5]. In Fig. 4, we see a fragment of this scheme with only five vertices numbered with 0–4. This fragment is created using the following formulae:

$$
\begin{aligned}
&\ell_0 = h(x_0),\\
&\ell_1 = h(x_1, \ell_0) \quad \ell_{01} = h(\ell_0, \ell_1)\\
&\ell_2 = h(x_2, \ell_{01})\\
&\ell_3 = h(x_3, \ell_{01}, \ell_2) \quad \ell_{23} = h(\ell_2, \ell_3) \quad \ell_{0123} = h(\ell_{01}, \ell_{23})\\
&\ell_4 = h(x_4, \ell_{0123}).
\end{aligned}
$$

To define this linking scheme in general case, we use binary codes to enumerate linking items. For example, we denote L_2 with `010`, i.e. with binary representation of 2. The non-leaf vertices are numbered by using additional symbol $*$. For example, ℓ_{23} is denoted as `01*` because it represents (is a parent-vertex of) a pair of leaves $\{\texttt{010}, \texttt{011}\}$. For similar reasons, ℓ_{0123} is represented by codeword

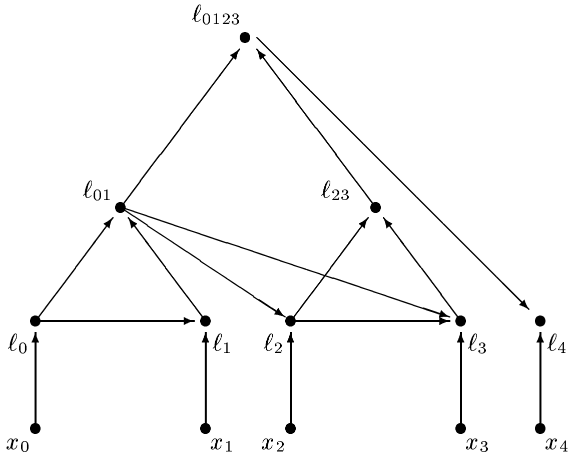

Fig. 4. The threaded-tree linking scheme [5].

$0**$ because it represents a pair of vertices $\{00*, 01*\}$. Let $b_{k-1}\dots b_0$ be binary representation of n. Then $\mathcal{H}_{n-1}$ is a set of codewords:

$$\mathcal{H}_{n-1} = \{b_{k-1}b_{k-2}\dots b_{i+1}0*\dots* \mid 0 \leq i < k,\ b_i = 1\}.$$

For example, in the scheme depicted in Fig. 4, we have $\mathcal{H}_3 = \{010, 00*\} = \{\ell_2, \ell_{01}\}$. Let $n' > n$ be another k-bit integer with binary representation $b'_{k-1}\dots b'_0$. Let m be the smallest index such that

$$b_{k-1} = b'_{k-1},\ b_{k-2} = b'_{k-2}, \dots, b_m = b'_m,$$

i.e. $b_{m-1} \neq b'_{m-1}$ and therefore, considering that $n' > n$, we know that $b_{m-1} = 0$ and $b'_{m-1} = 1$. In that case,

$$\mathcal{T}_{n,n'} = \mathcal{H}_{n'-1} \cup \{x_{n'}\} \cup \{b_{k-1}\dots b_m b_{m-1}\dots b_{i+1}\overline{b_i}*\dots* \mid 0 \leq i < m\}.$$

It can be easily proved that having two pairs

$$(\mathcal{H}_{m_1}, \mathcal{T}_{n_1,N}) \text{ and } (\mathcal{H}_{m_2}, \mathcal{T}_{n_2,N})$$

such that $n_1 \leq m_2 \leq N$ then $\mathcal{T}_{n_1,N}$ and $\mathcal{H}_{m_2}$ together are sufficient to compute a one-way link from L_{n_1} to L_{m_2}.