Stamp & Extend – Instant but Undeniable Timestamping based on Lazy Trees[*] [†]

Łukasz Krzywiecki Przemysław Kubiak
Mirosław Kutyłowski
Faculty of Fundamental Problems of Technology
Wrocław University of Technology
{lukasz.krzywiecki, przemyslaw.kubiak, miroslaw.kutylowski }@pwr.wroc.pl

Abstract

We present a Stamp&Extend time-stamping scheme based on linking via modified creation of Schnorr signatures. The scheme is based on lazy construction of a tree of signatures.

Stamp&Extend returns a timestamp immediately after the request, unlike the schemes based on the concept of timestamping rounds. Despite the fact that all timestamps are linearly linked, verification of a timestamp requires a logarithmic number of steps with respect to the chain length. An extra feature of the scheme is that any attempt to forge a timestamp by the Time Stamping Authority (TSA) results in revealing its secret key, providing an undeniable cryptographic evidence of misbehavior of TSA.

Breaking Stamp&Extend requires not only breaking Schnorr signatures, but to some extend also breaking Pedersen commitments.

Keywords: timestamping, undeniability, forgery evidence, Schnorr signature

1 Introduction

1.1 Legal Background

Timestamping is one of the very basic services that are necessary for electronic document flow. According to a recent legal definition [Eur12]:

> *'electronic time stamp' means data in electronic form which binds other electronic data to a particular time establishing evidence that these data existed at that time.*

Time stamps are perhaps as important as digital signatures for future applications: while a digital signature provides guarantees for document origin and its approval by the signatory, it does not prove when the signature was created. However, signing time is crucial for the legal consequences of a signed document. This concerns not only such trivial cases as documentation for online financial operations (e.g. on a stock exchange), but also administrative procedures, where a party participating in a procedure has a limited period of time to perform a legally valid action. Importance of timestamping was recognized in this legal environment – it was mentioned already by European Directive concerning electronic signature a decade ago [Eur00].

[*]The paper has been partially supported by the Polish Ministry of Science and Higher Education, grant N N206 369 839; the third author has been supported by the Foundation for Polish Science, "Mistrz" Programme.

[†]This is authors' version of the InTrust 2012 paper [KKK12]. The original publication is available at `www.springerlink.com`

Recently, European Commission proposed a new regulation concerning electronic identification and trust services [Eur12]. Time-stamping is one of the main elements of the proposed framework. The proposal states that electronic time-stamp *shall not be denied legal effect and admissibility as evidence in legal proceedings solely on the grounds that it is in electronic form.* In case of qualified time-stamps (i.e. the time-stamps issued by qualified service providers), *electronic time stamp shall enjoy a legal presumption of ensuring the time it indicates and the integrity of the data to which the time is bound.* Thereby, electronic time-stamps will become an important legal institution.

New laws concerning timestamping means facilitating use of electronic documents in legal proceedings. On the other hand, without a flawless technical framework electronic time-stamps may become a powerful tool for electronic frauds. This has not been so important so far, since it was not compulsory to accept electronic time-stamps in legal proceedings. Designing time-stamping systems must also take into account possibility of malicious activities of all participants of the process, including in particular *qualified service providers*, and the threat of breaking cryptographic schemes. Introducing a new technology into a legal framework should be coupled with effective e-forensics techniques. Possible legal conflicts concerning timestamps should be solvable on technical grounds. Otherwise, this technology can be used to create *time machines* in the legal framework. Namely, a party having backdoors to the timestamping services would have an opportunity to perfectly backdate electronic documents.

1.2 Certification and Secure Timestamping Devices

An idea which is dominant in existing implementations is to rely on special purpose *secure timestamping devices* - just like in the case of electronic signatures and *secure signature creation devices* holding private signing keys. As in case of signature creation devices, technical security and resistance to manipulations should be checked during certification process, where details are disclosed to trusted certification bodies. While certification process is a very important element of efforts for providing security, one has to keep in mind that it does not necessarily guarantee elimination of backdoors. Certification process reduces significantly the amount of design mistakes enforcing a rigorous design and implementation, as well as formulation of design goals and product features. However, this is only a process of checking of some properties against a certain list that may simply ignore or overlook some important issues.

Providing guarantees for security of *timestamping devices* is significantly harder than in case of *signature creation devices*. Indeed, in the last case the owner of the device has very strong reasons to protect it against attacks. Namely, compromising the signing key open doors to creation of perfect signatures on behalf of the attacked person. The situation might be very different for a TSA – it may attempt to retrieve the keys stored in the device in order to be able to backdate certain documents. We have to be aware of the fact that organized crime might be interested in controlling TSA in order to make perfect frauds efficiently.

The lessons learned from early e-voting implementations show that we should be very careful when entrusting blindly black box devices. Today, after disasters with e-voting machines in USA and Netherlands (which were due to blind trust to manufacturers and their products), it is much harder to convince the users to such solutions. There should be a strong argument behind the construction of timestamps that is not based solely on the assumption that the timestamping service provider and the manufacturer are honest and competent. Even if they are, there might be third parties that have better cryptographic and technical knowledge as well as better resources enabling them to exploit some unknown weaknesses of the devices.

1.3 Undeniable Timestamping – Related Work

1.3.1 Round Schemes.

The very first constructions of undeniable timestamps have been presented in [BdM91], [HS91]. The basic idea is simple: there is a service provider, TSA, that issues timestamps on demand. TSA builds a database secured with cryptographic means so that it allows append operation but does not allow insert operations and modifications of past records.

The basic structure here is a linear chain of hashes: the next element in the chain contains a signature of TSA not only on the digital data to be stamped, but also on the hash of the previous element in the chain. Therefore, it is impossible to change an element of the chain: any manipulation would change its hash value and therefore disconnect it from its successor. However, hash chains have a serious disadvantage. Since the hash chain is anchored at its initial element (confirmed by a third party or published), while checking a timestamp it is necessary to perform a number of operations that is linear in the number of timestamps. Obviously such a solution in not scalable and can be used only in small systems.

The idea to overcome this problem is to split time into rounds and link only data between the rounds. Within a round, TSA is executing a procedure that finally delivers a single value to be incorporated in a linear chain connecting the rounds. For making a round, different techniques has been proposed. One solution is to use one-way accumulators [BdM93]: all requests gathered during one round are accumulated into a single hash value, and at the end of the round each requester receives a compact proof (i.e., a timestamp) that her/his request is included in the hash value. The underlying hash function is build upon exponentiation modulo a product of two strong primes that have the same length. The scheme [BdM93] is a distributed one, with key generation procedure distributed as well (the latter is pretty complicated, see e.g., [DM10]), or alternatively, with centralized generation of the modulus (but then the central server must be trusted).

An alternative solution is to generate a single aggregated signature for all data submitted during a single round (see e.g., scheme [LBG08b] build upon bilinear maps).

However, majority of proposals use a kind of Merkle tree generated for the timestamped values located at leaves of the tree. This provides both undeniability as well as proofs of presence in the tree with a given root having logarithmic lengths. The idea is very simple, nevertheless there are some fine issues concerning the scheme:

- While it is obvious that a cryptographic hash function should be used to generate labels of the tree nodes, it is not completely clear which properties are really necessary. However, after formulating the requirements, the schemes can be examined via reduction proofs: finding a relationship between breaking a timestamping scheme and breaking some standard security assumption. For considerations of this kind see e.g. papers [BS04], [BLSW05],[BL06],[BN08],[BN10].

- It is not only hash function that matters – it turns out that carefully tailored construction of hash trees may improve efficiency of the scheme – see e.g. a design based on skewed trees [Lip02]. Moreover, trees are not necessarily the best data structure. Essentially, the same chain of trust can be built in a directed acyclic simple graph with a single sink – see e.g. a solution based on skip lists [BG06].

1.3.2 Instant Timestamping.

The main *drawback of the round approach* is improving scalability at the price of response time: instead of one-stop-shop there is an interaction between the requester and TSA extended until the end of the round. That is, *to obtain a timestamp the requester must wait for the end of the round.*

A notable exception among round-based schemes is the protocol from [LBG08a]: hashes of the requests are generated *in advance*, using chameleon hash function. Then a Merkle tree is built from the leaves being these hash-values produced in advance, and the root of the tree becomes the public commitment that can be used to authenticate the future timestamps. To avoid frauds the scheme from [LBG08a] distributes the trapdoor for the chameleon hash function among a few servers. Consequently, clients' requests are answered immediately, but operational costs of the service are increased by using a distributed protocol to generate preimages for the chameleon hashes, which is necessary to answer the requests.

1.4 Our Contribution

We change the approach presented in [LBG08a]. Instead from making commitments to the hashes of the future requests, the protocol presented below makes commitments to the randomness that will be used to generate signatures under answers to the requests. The crucial point is that if the same randomness is used to sign two different requests, then the private signature key leaks from TSA. Consequently, instead of designing a distributed system ([BdM93], [LBG08a]) we propose a cheaper centralized one, which is deterred from misbehavior by the threat of the signature key leak. The idea of deterrence by key leak was utilized in a stand-alone signature protocol [BKK10]. However, the scheme presented in [BKK10] is highly inefficient – a single signature must be composed of more than *eighty* Rabin-Williams sub-signatures.

Moreover, the commitments to the randomness are made gradually during the protocol execution, when the currently submitted requests are served. Since the randomness used for signature generation is independent from the hash values resulting from future requests, a separate setup phase for building a Merkle tree is no longer needed. For each timestamp generated, two commitments are prepared for a future use (i.e., a timestamp makes links to *future* nodes, and these links are independent from hash values resulting from the requests the links will be used for). In this way some binary tree is gradually created (cf. Fig. 2 on page 9), what shall be utilized by the timestamp verification procedure. The size of the tree is limited only by users' expectations to timestamp verification time, thus the size is not predetermined like in the scheme [LBG08a]. If size of the tree becomes too big, the private key material of TSA may be destroyed (see Subsect. 3.3) and a new tree could be initialized.

To sum up, we propose a Stamp&Extend time stamping scheme that has the following properties:

- the Time Stamping Authority is centralized, i.e., it can be run on a single server,
- interaction between the requester and the centralized TSA is limited to the request and its immediate response; there is no need to wait for the end of a round,
- every two timestamps issued by the same TSA are comparable with respect to the order they were requested (as in the simple linear linking scheme),
- communication complexity from TSA to the requester is logarithmic in the number of timestamps issued so far, and at the same time computational complexity for issuing a timestamp is constant,
- issuing a backdated document exposes the signing key of TSA,
- the main security features are based on hardness of discrete logarithm problem and on non-repudiation of Schnorr signatures.

1.5 Security Requirements

To correctly describe the idea of deterrence mentioned in Subsect. 1.4 we shall first explain what forgery means in case of the Stamp&Extend scheme.

The scheme is build upon the classical protocol, i.e., upon the linear chain of hashes: the verification procedure (see Algorithm 4) checks if the timestamp verified has the link to its predecessor in the chain. If the link is absent the timestamp is rejected and TSA is caught cheating (TSA signs each timestamp generated). Ill-formedness of a timestamp is not penalized in our scheme by TSA's private key leak, however any later efforts of TSA to replace the ill-formed timestamp *are* penalized.

As mentioned in Subsect. 1.4, we focus on TSA signatures: the linear chain structure is augmented with a system of commitments that are commitments to randomness used in the signatures. Each commitment corresponds to a single position in the linear chain of timestamps, and the system of commitments has the form of a binary tree (such a form accelerates verification of a single commitment). The two structures (the linear chain and the binary tree) are fused together – see Fig. 2 on page 9, and each commitment in fact *determines* the in-chain position of a timestamp, for whose signature the committed randomness is used for. Therefore, even if some positions are left empty by the TSA, i.e., even if the malicious authority makes a gap for a future use and starts a new chain on some forward position, the new and the old chain are glued together by the binary structure of commitments. Hence the term *alleged chain* is justified and may be used in the following definition:

Definition 1.1 *By a forgery we mean issuing two timestamps for the same position of the alleged chain.*

The Stamp & Extend scheme must ensure that:

1. The system of commitments does not weaken the signature scheme used by TSA.

2. The deterrence mechanism is efficient, i.e. in case of a forgery the private signature key of TSA leaks with a non-negligible probability.

Condition 1 is addressed by considering external adversaries – cf. Definition 3.3 and Theorem 3.4 in Subsect. 3.1. Condition 2 is addressed by Theorem 3.5 from Subsect. 3.2.

2 Stamp & Extend Protocol

2.1 Protocol's Building Blocks

2.1.1 Schnorr Signatures.

Our construction is based on Schnorr signatures [Sch91b], [BSI12, Subsect.4.2.3]. Let us recall the scheme briefly. Let G be a group of a prime order q for which Discrete Logarithm Problem (DLP) is hard. Let g be a fixed generator of this group.

Private key of a user u is a number $x \in [1, q-1]$ chosen at random. The corresponding public key is an element of group G computed as $y := g^x$.

For signature creation we use a cryptographic hash function H. A signature of a message M is created as follows:

1. the signer chooses an integer $k \in [1, q-1]$ uniformly at random,
2. $r := g^k$,
3. $e := H(M||r)$ ($||$ stands for concatenation),
4. $s := (k - xe) \bmod q$,
5. output signature (e, s).

Verification of this signature is performed as follows:

1. $r' := g^s \cdot y^e$,
2. $e' := H(M||r')$,
3. if $e = e'$ then return `true`, otherwise return `false`.

The Schnorr signature scheme is very elegant and simple. The main disadvantage of Schnorr signatures was a US patent [Sch91a], which encouraged manufacturers to implement other patent-free algorithms based on DLP. This situation has changed, as the patent [Sch91a] expired in 2008. For smart cards there is another disadvantage: the computation of r has to be done on the card before the message M is hashed. As hashing is executed typically outside the card, this increases the number of messages exchanged with the card. In our case this is not a problem, as we aim to create Schnorr signatures on a server.

2.1.2 Pedersen Commitments [Ped91].

Let G be group of prime order q for which DLP is hard. Let g, h be generators of G such that $\log_g h$ is not known to anybody. Commitment c to k is obtained by choosing $\ell \in \{0, 1, \ldots, q-1\}$ uniformly at random and assigning:

$$c := g^k \cdot h^\ell.$$

The commitment can be later opened by the commiter by revealing k and ℓ. Commitment c reveals no information about k. Moreover, opening the commitment c to a k' such that $k' \neq k$ reveals the discrete logarithm $\log_g h$. Therefore it is infeasible to replace k by k'.

2.2 The Protocol

2.2.1 Initialization.

When starting a TSA the first step to do is to generate its public and private parameters. Before we do it, we have to choose a group G of a prime order, say q, where DLP is hard. We shall also use a secure hash function H. Setup of keys for TSA is depicted by Algorithm 1 .

```
Actors : TSA
Input  : group $G$ of prime order $q$, generators $g, h$ of $G$ such that $\log_g h$ is not
         known to anybody
Output : public key $y$, private key $x$, pair of starting parameters $(k_1, \ell_1)$, a
         Pedersen commitment $c_1$
1 for $(k_1, \ell_1)$ begin
2     choose $x \in \{1, 2, \ldots, q-1\}$ uniformly at random
3     $y := g^x$
4     return $x$ as the private signing key
5     return and export $y$ as the public signing key
6     choose $k_1, \ell_1 \in \{0, 1, \ldots, q-1\}$ uniformly at random
7     $c_1 := g^{k_1} h^{\ell_1}$
8     return $(k_1, \ell_1)$ as the first pair of private timestamping parameters
9     return and export $c_1$ as the public commitment for the first timestamp
```

Algorithm 1: Setup of keys for TSA.

After the keys are generated, it is necessary to confirm public parameters by issuing certificate(s) $Cert$ (see Algorithm 2). A standard form of publishing $Cert$ has to be chosen (e.g. publishing in a newspaper) so that the keys cannot be changed afterwards.

2.2.2 Creating timestamps.

TSA stores the following data during protocol execution:

- the index of the last timestamp issued $i - 1$,
- a private list P of pairs of exponents $[(k_i, \ell_i), \ldots, (k_{2i-1}, \ell_{2i-1})]$

- a public file C with the list of Pedersen commitments $[c_1, \ldots, c_{2i-1}]$,
- a public file HS that includes an initial value HS_0 (which is the certificate of TSA) and elements $\mathrm{HS}_j = (T_j, \mathrm{H}_j)$ for $j = 1, \ldots, i-1$, where T_j is the jth timestamp and H_j is the corresponding timestamped value (i.e. a hash value together with identifier of the hash function used [Kal02]).

```
Actors : TSA, Certification Authority (CA)
Input  : public signing key $y$, initial commitment $c_1$
Output : a certificate $Cert$ for $y$ and $c_1$
1 begin
2     CA generates a digital certificate $Cert$ for $y$ and $c_1$ in a standard way
3     CA publishes $Cert$
4     TSA initializes HS as the list $[\mathrm{HS}_0]$, where $\mathrm{HS}_0 = Cert$
```

Algorithm 2: Generating certificates for TSA.

Creation of the ith timestamp (see Algorithm 3) requires using c_i as well as *issuing two additional commitments* c_{2i}, c_{2i+1} and storing them in C. The list $C = [c_1, c_2, c_3, c_4, \ldots, c_7, \ldots, c_{2i}, c_{2i+1}]$ may be perceived as a list of consecutive layers of a binary tree (the currently bottom layer may not be completed):

$$C = [\{c_1\}, \{c_2, c_3\}, \{c_4, \ldots, c_7\}, \ldots, \{c_{2 \cdot 2^{\lfloor \log_2 i \rfloor}}, \ldots, c_{2i}, c_{2i+1}\}]$$

By making a signature of record (1), which contains the values c_{2i}, c_{2i+1}, TSA creates an authentication link from c_i to the nodes c_{2i}, c_{2i+1} (note that c_i is a commitment to the randomness used in the signature of (1)). The sequence of records (2) is in fact a kind of authentication path from the node c_i to the root c_1. The procedure is additionally illustrated by Fig. 1 on page 7 .

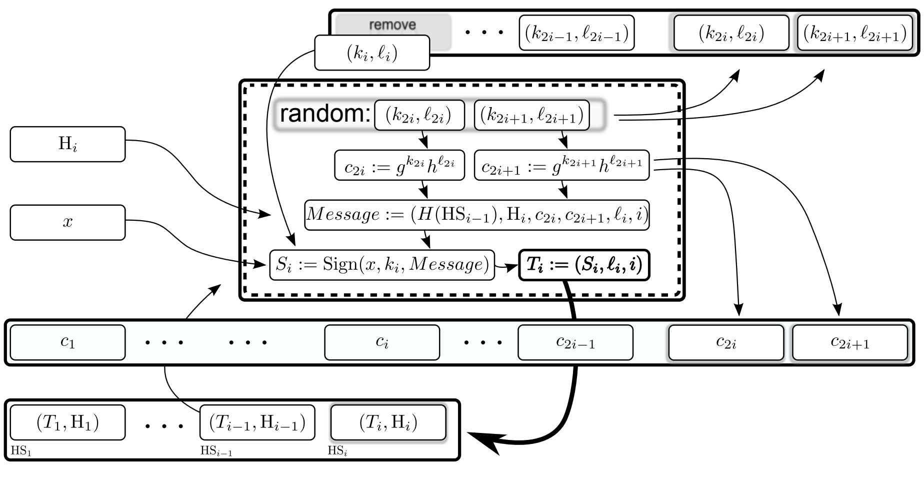
remove
(k_{2i-1}, ℓ_{2i-1})
(k_{2i}, ℓ_{2i})
(k_{2i+1}, ℓ_{2i+1})
(k_i, ℓ_i)
random:
(k_{2i}, ℓ_{2i})
(k_{2i+1}, ℓ_{2i+1})
H_i
$c_{2i} := g^{k_{2i}} h^{\ell_{2i}}$
$c_{2i+1} := g^{k_{2i+1}} h^{\ell_{2i+1}}$
x
$Message := (H(\mathrm{HS}_{i-1}), \mathrm{H}_i, c_{2i}, c_{2i+1}, \ell_i, i)$
$S_i := \mathrm{Sign}(x, k_i, Message)$
$T_i := (S_i, \ell_i, i)$
c_1
c_i
c_{2i-1}
c_{2i}
c_{2i+1}
(T_1, H_1)
$(T_{i-1}, \mathrm{H}_{i-1})$
(T_i, H_i)
HS_1
HS_{i-1}
HS_i

Figure 1: Creation of the i-th timestamp.

2.2.3 Checking timestamps.

Timestamp verification can be executed according to Algorithm 4 given on page 9 .

Note that immediate access to C and real-time execution of line 8 of Algorithm 4 is not necessary. If the requester of a timestamp receives a sequence of records (2) for $j = \lfloor i/2^\alpha \rfloor$, where $\alpha = 0, 1, \ldots, \lfloor \log_2 i \rfloor$, and all signatures S_j are correct, then these signatures contain a kind of declaration to corresponding commitments c_j available to the public. The declarations are $c'_j = g^{e_j} y^{s_j} h^{\ell_j}$ – note that ℓ_j as well as indexes j are included in the signed data of the corresponding records (1). Accordingly, if TSA is

```
Actors : TSA, a requester
Input :
  • index $i-1$ of the last timestamp issued
  • list $C = [c_1, \dots, c_{2i-1}]$ of Pedersen commitments
  • list $P = [(k_i, \ell_i), \dots, (k_{2i-1}, \ell_{2i-1})]$ of pairs of unused private exponents
  • list HS including $\mathrm{HS}_0$ and elements $\mathrm{HS}_j = (T_j, \mathrm{H}_j)$ for $j = 1, \dots, i-1$
  • value $\mathrm{H}_i$ to be timestamped delivered by the requester
Output : updated: index $i$ and lists $C$, $P$, HS
1  begin
2      choose $k_{2i}, \ell_{2i}, k_{2i+1}, \ell_{2i+1} \in \{0, 1, \dots, q-1\}$ uniformly at random
3      $c_{2i} := g^{k_{2i}} h^{\ell_{2i}}, \quad c_{2i+1} := g^{k_{2i+1}} h^{\ell_{2i+1}}$
4      append $c_{2i}, c_{2i+1}$ to $C$
5      $k := k_i$
6      remove $(k_i, \ell_i)$ from $P$, append $(k_{2i}, \ell_{2i}), (k_{2i+1}, \ell_{2i+1})$ to $P$
7      create Schnorr signature $S_i$ using random parameter $k$ from line 5 for the
       following "message":

           $(H(\mathrm{HS}_{i-1}), \mathrm{H}_i, c_{2i}, c_{2i+1}, \ell_i, i),$        (1)

       where $H()$ is cryptographically secure hash function used by TSA
8      define the $i$th timestamp as $T_i := (S_i, \ell_i, i)$    /* note that one of
       the exponents for commitment $c_i$ is revealed in the
       timestamp */
9      append $(T_i, \mathrm{H}_i)$ to list HS as the last element $\mathrm{HS}_i$    /* note that HS
       and $C$ allow to reconstruct completely the message
       (1) */
10     return index $i$ to the requester
11     alternatively: return the following sequence of records to the requester

           $(S_j, H(\mathrm{HS}_{j-1}), \mathrm{H}_j, c_{2j}, c_{2j+1}, \ell_j, j)$        (2)

       for $j = \lfloor i/2^\alpha \rfloor$, where $\alpha = 0, 1, \dots, \lfloor \log_2 i \rfloor$.
```

Algorithm 3: Creating a timestamp by TSA.

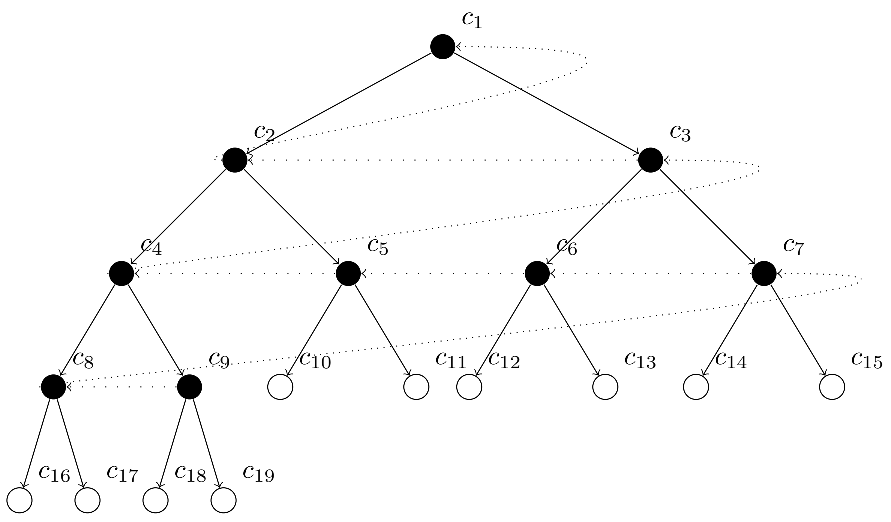

Figure 2: The fuse of two structures (a chain and a binary tree) generated by consecutive calls of Algorithm 3 . The binary tree reflects dependencies between timestamps and commitments. The filled circles denote timestamps (and also the commitments already used), the empty circles are unused commitments. The dotted arrows denote hash links between consecutive timestamps in the chain.

```
Actors : the requester or a third party
Input  : index $i$ and access to the public lists $C$ and HS
         or the sequence of records (2) and certificate $Cert$.
Output   correct or incorrect with a proof of inconsistency
:
1  begin
2      check certificate $Cert$ and retrieve public key $y$ from it
3      for $\alpha = 0, \ldots, \lfloor \log_2 i \rfloor$ do
4          $j := \lfloor i/2^{\alpha} \rfloor$
5          verify the signature $S_j = (e_j, s_j)$ under the corresponding "message"
           reconstructed from record (2) that includes $S_j$
6          check if the second element of the record (2) is indeed a hash of $\mathrm{HS}_{j-1}$
7          reconstruct $c_j$ by computing $c_j := g^{e_j} y^{s_j} h^{\ell_j}$
8          compare $c_j$ with the $j$th element of $C$
9          for $j > 1$ check if $c_j$ appears in the record (2) corresponding to $\alpha + 1$:
10         case if $j$ is odd, then this should be the second element
11             if $j$ is even, then this should be the first element
12         for $j = 1$ check if $c_1$ is confirmed by $Cert$
```

Algorithm 4: Verification of a timestamp.

cheating and records (2) contain at least one false declaration to value c_j, then it would be provably evident due to corresponding signature S_j: Note that if the sequence of declarations

$$c'_i, c'_{\lfloor i/2 \rfloor}, \ldots, c'_{\lfloor i/2^{\lfloor \log_2 i \rfloor - 2} \rfloor}, c'_{\lfloor i/2^{\lfloor \log_2 i \rfloor - 1} \rfloor}, c_1$$

is different from the publicly available sequence of commitments

$$c_i, c_{\lfloor i/2 \rfloor}, \ldots, c_{\lfloor i/2^{\lfloor \log_2 i \rfloor - 2} \rfloor}, c_{\lfloor i/2^{\lfloor \log_2 i \rfloor - 1} \rfloor}, c_1$$

then there is some index for which the sequences differ. By β let us denote the first such index from the right. That is, $c_\beta \neq c'_\beta$, but $c_{\lfloor \beta/2 \rfloor} = c'_{\lfloor \beta/2 \rfloor}$ (at worst $\lfloor \beta/2 \rfloor = 1$). This means that the corresponding "messages" (1) for $i = \lfloor \beta/2 \rfloor$ are different, because $c_\beta \neq c'_\beta$, but the randomness used to make the signatures under the "messages" is the same, because $c_{\lfloor \beta/2 \rfloor} = c'_{\lfloor \beta/2 \rfloor}$. Assuming that Schnorr signatures are hard to repudiate this leads to leakage of signature keys.

See that due to hash value $H(\mathrm{HS}_{i-1})$ present in "message" (1) the Stamp & Extend scheme may indeed be considered as a simple linear chain augmented with a mechanism providing logarithmic shortcuts to the root of the chain. The verification from Algorithm 4 (including line 8 of the Algorithm) executed for a given timestamp does not prove integrity of the whole data. It only shows that the timestamp checked must stand on the claimed position in the chain of timestamps, because the timestamp is anchored by a particular path (i.e., by a particular shortcut) to the certificate $Cert$. Of course, it witnesses that some parts of file HS are correct. Moreover, anybody can access the information from C and HS and check integrity of timestamps starting at any possible location[1]. Even possibility of such an external audit (e.g., executed by competitors of TSA), should strongly discourage any misbehavior by TSA.

3 Security Analysis

3.1 External Adversaries $\mathcal{A}$

To facilitate security analysis let us first consider a restricted model of an external adversary (the restricted model shall be extended in Def. 3.3): Assume an adversary that attempts to create a timestamp matching one of the published commitments.

Definition 3.1 *$\mathcal{A}_1$ is given access to:*

- *list C of commitments (initially $C = [c_1]$),*
- *remaining data from* HS *(initially* $\mathrm{HS} = [Cert]$*).*

$\mathcal{A}_1$ can request TSA to issue some number of timestamps for hashes of $\mathcal{A}_1$'s choice. According to the number of requests (which is bounded by a polynomial in the security parameter $\kappa = \log_2 q$) both the list C and the file HS *will grow.*

We assume that $\mathcal{A}_1$ wins, if he manages to issue a valid triple (S'_j, ℓ'_j, j), where S'_j is any[2] *Schnorr signature being verified with TSA's public key y, corresponding to one of the unused commitments c_j.*

Theorem 3.2 *In the standard model, the advantage of $\mathcal{A}_1$ is negligible, provided that solving DLP in $\langle g \rangle$ succeeds with a negligible probability.*

[1]For example, each requester receiving a timestamp (i.e., each client application) may always verify *a constant* number n_{ver} of timestamps: the one received and $n_{ver} - 1$ consecutive predecessors of a randomly chosen timestamp in the chain (the random choice is made by the requester). Moreover, we may assume that a local copy of all timestamps received is maintained by the requester, and a locally stored timestamp is compared with the newly received one if both are on the same position in the hash chain. Note that in case of misbehavior TSA may try to reduce probability of its detection by issuing an enormous number of timestamps for itself. However, this looks suspicious and may be the reason for thorough inspection of TSA.

[2]That is, we do not require that the message "signed" with S'_j has the form of a timestamp. Consequently, task of the adversary $\mathcal{A}_1$ does not become harder.

Proof Let us assume that F_1 is an algorithm run by $\mathcal{A}_1$ that for n unused commitments enables him to deliver a valid triple (S'_j, ℓ'_j, j). Let $\delta_1^{(n)}$ denote success probability of F_1. Based on this we construct an algorithm (see Algorithm 5) that attempts to solve discrete logarithm problem for b, that is, to find a such that $b = g^a$.

```
Input  : a finite group generated by element $g$
         an element $b \in \langle g \rangle$ for which discrete logarithm $a$ is sought
Output : discrete logarithm $a$ of $b$ or $\perp$ in case of failure
1  begin
2      assign $h := b$
3      generate uniformly at random a “secret” key $x \in \{1, 2, \ldots, q-1\}$ of TSA
       and public key $y = g^x$
4      choose $k_1, \ell_1 \in \{0, 1, \ldots, q-1\}$ uniformly at random
5      generate $c_1 = g^{k_1} h^{\ell_1}$
6      run $F_1$
7      $F_1$ chooses some $n$ bounded by a polynomial in $\kappa$: $n \leq Poly(\kappa)$
8      generate timestamps for $\mathrm{H}_i$ requested by $F_1$, $i = 1, \ldots, n$, (accordingly, file
       HS and list $C = [c_1, \ldots, c_{2n+1}]$ is made, where $c_i = g^{k_i} h^{\ell_i}$ for uniformly
       chosen $k_i, \ell_i \in \{0, 1, \ldots, q-1\}$, $i = 2, \ldots, 2n+1$)
9      with probability $\delta_1^{(n)}$ algorithm $F_1$ returns some valid triple $(S'_j, \ell'_j, j)$ where
       $j \in \{n+1, \ldots, 2n+1\}$, where $S'_j = (e'_j, s'_j)$ is *any* Schnorr signature
       being verified with TSA’s public key $y$, signature made for message $m'_j$
       adaptively chosen by $F_1$
10     generate timestamps for $i = n+1, \ldots, 2n+1$ for $\mathrm{H}_i$ being hashes of
       random messages, as a result in the $j$th timestamp a triple $(S_j, \ell_j, j)$ is
       generated, where $S_j = (e_j, s_j)$
11     if $\ell_j \neq \ell'_j \bmod q$ then
12         $a := (s_j - s'_j + x(e_j - e'_j)) \cdot (\ell'_j - \ell_j)^{-1} \bmod q$
13         return $a$
14
15     return $\perp$
```

Algorithm 5: Breaking DLP with F_1.

Note that the assignment from line 12 for computing a follows from the equality $c_j = g^{s_j} y^{e_j} h^{\ell_j} = g^{s'_j} y^{e'_j} h^{\ell'_j}$. As ℓ_j is chosen at random, the condition from line 11 is false with probability $\frac{1}{q}$, thus Algorithm 5 returns a with probability $\delta_1^{(n)} \cdot (1 - \frac{1}{q})$. If breaking DLP has negligible success probability, then $\delta_1^{(n)} \cdot (1 - \frac{1}{q})$ must be negligible, too. As $1 - \frac{1}{q}$ is very close to 1, the value $\delta_1^{(n)}$ must be negligible, too. ∎

Let us now extend the first model of external adversary: $\mathcal{A}_2$ is given access to the same data as $\mathcal{A}_1$, but is not restricted to utilize unused commitments only:

Definition 3.3 *$\mathcal{A}_2$ is given access to:*

- *list C of commitments (initially $C = [c_1]$),*
- *remaining data from* HS *(initially* HS $= [Cert]$*).*

$\mathcal{A}_2$ can request TSA to issue some number of timestamps for hashes of $\mathcal{A}_2$'s choice.

We assume that $\mathcal{A}_2$ wins, if he manages to issue a valid triple (S'_j, ℓ'_j, j), where S'_j is any *Schnorr signature being verified with TSA's public key y, corresponding to* any *commitment c_j.*

Theorem 3.4 *In the random oracle model, the advantage of $\mathcal{A}_2$ is negligible, provided that any adaptive chosen message attack aimed at private key of Schnorr signatures succeeds with a negligible probability.*

Sketch Let us assume that F_2 is an algorithm run by $\mathcal{A}_2$ that for $2n+1$ commitments enables him to deliver a valid triple (S'_j, ℓ'_j, j). By

$$\delta^{(n)}_{eq,usd}, \delta^{(n)}_{neq,usd}, \delta^{(n)}_{eq,unsd}, \delta^{(n)}_{neq,unsd}$$

we denote the probabilities that F_2 succeeds respectively under the following conditions:

$$\begin{aligned}
&(\ell'_j = \ell_j) \wedge (1 \leq j \leq n),\\
&(\ell'_j \neq \ell_j) \wedge (1 \leq j \leq n),\\
&(\ell'_j = \ell_j) \wedge (n+1 \leq j \leq 2n+1),\\
&(\ell'_j \neq \ell_j) \wedge (n+1 \leq j \leq 2n+1).
\end{aligned}$$

Let $\delta^{(n)}_2 = \delta^{(n)}_{eq,usd} + \delta^{(n)}_{neq,usd} + \delta^{(n)}_{eq,unsd} + \delta^{(n)}_{neq,unsd}$.

On top of algorithm F_2 we build two algorithms, both incorporated into a single procedure (in fact the procedure defines the Challenger). On the very beginning of the resulting procedure a symmetric coin is tossed and according to toss result one of the two algorithms is executed. From F_2's point of view both algorithms are indistinguishable, that is, F_2 does not "know", what have called it. The coin toss shall introduce factor $\frac{1}{2}$ to reduction tightness.

We assume that F_2 can query q_H times random oracle $O_H(\cdot)$ for hash function H, and can query q_{sig} times random oracle $O_{sig,y}$, where $y \in \langle g \rangle$ is a public key for verification of signatures made by $O_{sig,y}$. Each of q_H, q_{sig} is bounded by some polynomial in a security parameter κ. Let len_H denote bit length of H.

We assume that all calls of the oracles made by F_2 shall be intercepted by the procedure mentioned above. The procedure shall compare arguments of the call with records of table T_H that were inserted by the second of the two algorithms, and if necessary a result from the appropriate table is returned to F_2.

The first of the two algorithms chosen by the coin toss works analogously to Algorithm 5: each occurrence of F_1 in Algorithm 5 should be replaced by F_2 and in line 9 of the algorithm $\delta^{(n)}_1$ should be replaced by $\delta^{(n)}_2$ and "$j \in \{n+1, \ldots, 2n+1\}$" by "$j \in \{1, \ldots, 2n+1\}$". Of course the modification is aimed at the event $\ell'_j \neq \ell_j$ which occurs with probability $\delta^{(n)}_{neq,usd} + \delta^{(n)}_{neq,unsd}$. Hence counting the coin toss, that is the probability that the algorithm resulting from the modification will fit the event, we get that the instance of DLP is solved with probability equal to $\frac{1}{2}(\delta^{(n)}_{neq,usd} + \delta^{(n)}_{neq,unsd})$. Therefore we get

$$\delta^{(n)}_{neq,usd} + \delta^{(n)}_{neq,unsd} \leq 2\epsilon^{(2n+1)}, \tag{3}$$

where $\epsilon^{(k)}$ is the upper bound for probability that DLP in group $\langle g \rangle$ may be broken with effort proportional to k.

The second algorithm is the Algorithm 6 . We shall argue that in the random oracle model its advantage differs negligibly from the advantage of an adaptive chosen message attack on private key of Schnorr signature scheme.

At the beginning of Algorithm 6 the target public key y is masked and randomized: $\tilde{y} := y \cdot g^{\tilde{x}}$. Thus, from F_2's point of view, the resulting public key $\tilde{y}$ of the time stamping service is uniformly distributed in $\langle g \rangle$. However, it is easy to convert the signatures obtained from the oracle $O_{sig,y}(\cdot)$ for public key y to signatures for public key $\tilde{y}$. It requires only "shifting" the second component of the signature (see the line 8).

Algorithm 6 makes commitments c_i in a different way than during the actual execution of Stamp&Extend. Namely, they are derived from signatures obtained from oracle $O_{sig,y}(\cdot)$ for random messages. The random messages are chosen from domain corresponding to records (1) – set $\mathcal{MS}$ defined in line 5 of Algorithm 6 describes the first five coordinates of this message space, the last coordinate is message's index corresponding to the index of commitment currently made. The purpose of submitting random messages to the oracle is separation of the main operations during creation of Schnorr signatures by TSA. Namely, the value r is calculated *before* the message to

```
Input  : a finite group generated by element $g$
         public key $y \in \langle g \rangle$ of Schnorr signatures
         access to hashing oracle $O_H(\cdot)$ and to signing oracle $O_{sig,y}(\cdot)$
Output : discrete logarithm $x$ of $y$ or $\perp$ in case of failure
1  begin
2      choose $\tilde{x} \in \{0, 1, \ldots, q-1\}$ uniformly at random
3      set public key of the timestamping service as $\tilde{y} = y \cdot g^{\tilde{x}}$    /* masking $y$ */
4      choose $h \in \langle g \rangle$ at random
5      set
       $\mathcal{MS} := \{0, \ldots, 2^{len_H}-1\} \times \{0, \ldots, 2^{len_H}-1\} \times \langle g \rangle \times \langle g \rangle \times \{0, \ldots, q-1\}$
        /* message space for the first 5 components of (1) */
6      choose
       $m_1 \in \{O_H(\mathrm{HS}_0)\} \times \{0, \ldots, 2^{len_H}-1\} \times \langle g \rangle \times \langle g \rangle \times \{0, \ldots, q-1\} \times \{1\}$
       with uniform probability distribution
7      call oracle $O_{sig,y}(\cdot)$ for $m_1$, let $(e_1, s_1)$ be the signature returned for
       $O_{sig,y}(m_1)$
8      set $(\widetilde{e_1}, \widetilde{s_1}) := (e_1, s_1 - \tilde{x}e_1)$
9      assign $r_1 := g^{s_1}y^{e_1}(= g^{\widetilde{s_1}}\tilde{y}^{\widetilde{e_1}})$ and choose $\ell_1 \in \{0, 1, \ldots, q-1\}$ uniformly
       at random
10     put entry $(1, m_1||r_1, \mathrm{NULL}, \widetilde{e_1})$ to $T_H$ and entry $(1, \widetilde{e_1}, \widetilde{s_1}, \ell_1)$ to $T_{sig}$
11     publish $c_1 = r_1 \cdot h^{\ell_1}$ as the first commitment on list $C$
12     run $F_2$
13     $F_2$ chooses some $n$ bounded by a polynomial in $\kappa$: $n \leq Poly(\kappa)$
14     for each timestamp request $\mathrm{H}_i$, $i = 1, \ldots, n$, from $F_2$ do
15         choose $m_{2i} \in \mathcal{MS} \times \{2i\}$, $m_{2i+1} \in \mathcal{MS} \times \{2i+1\}$ with uniform
           probability distribution
16         let $(e_{2i}, s_{2i})$ be result of $O_{sig,y}(m_{2i})$, $(e_{2i+1}, s_{2i+1})$ be result of
           $O_{sig,y}(m_{2i+1})$
17         set $(\widetilde{e_{2i}}, \widetilde{s_{2i}}) := (e_{2i}, s_{2i} - \tilde{x}e_{2i})$,
           $(\widetilde{e_{2i+1}}, \widetilde{s_{2i+1}}) := (e_{2i+1}, s_{2i+1} - \tilde{x}e_{2i+1})$
18         set $r_{2i} := g^{\widetilde{s_{2i}}}\tilde{y}^{\widetilde{e_{2i}}}$ and choose $\ell_{2i} \in \{0, 1, \ldots, q-1\}$ uniformly at
           random
19         put entry $(2i, m_{2i}||r_{2i}, \mathrm{NULL}, \widetilde{e_{2i}})$ to $T_H$ and entry $(2i, \widetilde{e_{2i}}, \widetilde{s_{2i}}, \ell_{2i})$ to
           $T_{sig}$
20         set $r_{2i+1} := g^{\widetilde{s_{2i+1}}}\tilde{y}^{\widetilde{e_{2i+1}}}$ and choose $\ell_{2i+1} \in \{0, 1, \ldots, q-1\}$
           uniformly at random
21         put entry $(2i+1, m_{2i+1}||r_{2i+1}, \mathrm{NULL}, \widetilde{e_{2i+1}})$ to $T_H$ and entry
           $(2i+1, \widetilde{e_{2i+1}}, \widetilde{s_{2i+1}}, \ell_{2i+1})$ to $T_{sig}$
22         attach $c_{2i} = r_{2i} \cdot h^{\ell_{2i}}$ and $c_{2i+1} = r_{2i+1} \cdot h^{\ell_{2i+1}}$ to the public list $C$
23         set $\widetilde{m_i} := (O_H(\mathrm{HS}_{i-1}), \mathrm{H}_i, c_{2i}, c_{2i+1}, \ell_i, i)$
24         if $O_H(\widetilde{m_i}||r_i)$ has been called by $F_2$ earlier and $O_H(\widetilde{m_i}||r_i) \neq \widetilde{e_i}$ then
25             return $\perp$
26         replace $(i, m_i||r_i, \mathrm{NULL}, \widetilde{e_i})$ in $T_H$ with $(i, m_i||r_i, \widetilde{m_i}||r_i, \widetilde{e_i})$
           /* hash replacement */
27         return to $F_2$: the record $\widetilde{m_i}$ and the signature $(\widetilde{e_i}, \widetilde{s_i})$ read from $T_{sig}$
28     with probability $\delta_2^{(n)}$ algorithm $F_2$ returns some valid triple $(S'_j, \ell'_j, j)$,
       where $j \in \{1, \ldots, 2n+1\}$ and $S'_j = (e'_j, s'_j)$ is any Schnorr signature being
       verified with TSA's public key $\tilde{y}$; the Schnorr signature is made for message
       $m'_j$ adaptively chosen by $F_2$
29     if $\ell_j = \ell'_j \bmod q$ and $\widetilde{e_j} \neq e'_j \bmod q$ then
30         $x := (s'_j - \widetilde{s_j}) \cdot (\widetilde{e_j} - e'_j)^{-1} - \tilde{x} \bmod q$    /* based on
                        $k_j = s'_j + (x+\tilde{x})e'_j = \widetilde{s_j} + (x+\tilde{x})\widetilde{e_j} \bmod q$ */
31
32         return $x$
33
34     return $\perp$
```

Algorithm 6: Breaking private key of Schnorr signatures with F_2.

be signed is presented by F_2. We have to keep in mind that the oracle $O_{sig,y}(\cdot)$ is a black box, and its operations are inseparable. So we cannot otherwise get the parameter $r = g^k$ which indirectly appears in the commitment $c = g^k h^l$.

At this point we take advantage of the random oracle model for the hash function H and of that the whole communication between F_2 and the oracles is intercepted by the main procedure defining the Challenger. Accordingly, the Challenger may cheat algorithm F_2 and return some values of its choice instead of the values that would be calculated by the oracles. In our case this happens with values returned for arguments $\widetilde{m_i}$ in line 23 of Algorithm 6 . That is, commitments are made on basis of signatures made for the random messages m_i, and then Algorithm 6 cheats algorithm F_2 that the signature that was created for m_i is in fact a fresh signature for $\widetilde{m_i}$, the signature that utilizes randomness r_i committed in c_i. To make the cheating successful the Challenger must intercept all future calls of $O_H(\cdot)$ for argument $\widetilde{m_i}||r_i$ and return the answer $\widetilde{e_i}$, as it would be the answer of oracle $O_H(\cdot)$ called by the signature algorithm. Therefore Algorithm 6 makes at most n artificial collisions on H ("at most" because some of the collisions could also appear if the oracle $O_H(\cdot)$ would be used for the arguments).

The cheating would not be successful if at least one of the arguments $\widetilde{m_i}||r_i$, $i = 1, \ldots, n$, was submitted by F_2 to the oracle $O_H(\cdot)$ before the argument was assembled by Algorithm 6 , and the result returned by $O_H(\cdot)$ was different than the result needed by Algorithm 6 (cf. condition in line 24). The cheating should be successful if F_2 would not detect any of the artificial collisions. Probability of finding some of the artificial collisions could be upper-bounded by probability of finding any collision on H. Note that F_2 knows at most $q_H + q_{sig} + 2n$ results of $H(\cdot)$ (indeed, q_{sig} hash values are due to signatures returned by the oracle $O_{sig,y}(\cdot)$ on F_2's requests, n results are returned by the Challenger as $O_H(\mathrm{HS}_{i-1})$, and n results are $\widetilde{e_i} = e_i$ which are also delivered by the Challenger). Being pretty conservative in estimating the upper bound for collision detection by F_2 for the purpose of this estimation we shall treat oracle $O_H(\cdot)$ just like a function (denoted as $H(\cdot)$). Let us count the values of $H(\cdot)$ that could be not used now because of the artificial collisions: in the worst case $\widetilde{m_i}||r_i$ could have been the only argument yielding $H(\widetilde{m_i}||r_i)$, in such a case replacing $H(\widetilde{m_i}||r_i)$ with $H(m_i||r_i)$, $i = 1, \ldots, n$, would limit results' space by at most n elements. Therefore the upper bound for collision detection by F_2 is $(q_H + q_{sig} + 2n) \cdot (q_H + q_{sig} + 2n - 1)/(2(2^{len_H} - n))$.

Since e_j, e'_j are both returned by the random oracle (value e'_j is always returned by $O_H(\cdot)$ when S'_j is being verified, and we have assumed that signature S'_j is valid), the event "$\widetilde{e_j} = e'_j \bmod q$" occurs with probability at most $2(q_H + q_{sig} + 2n - 1)/\min(q, 2^{len_H})$ (the coefficient "2" in the numerator results from the upper bound for maximum unbalance of distribution of results modulo q reduction, $2n - 1$ is the number of hash values obtained apart from $\widetilde{e_j}$ from the Challenger).

Finally, the attack is successful if Challenger's cheating is not detected and the condition in line 29 is satisfied. The probability of a successful attack is then at least:

$$(\delta^{(n)}_{eq,usd} + \delta^{(n)}_{eq,unsd}) \cdot \left(1 - \frac{(q_H+q_{sig}+2n)\cdot(q_H+q_{sig}+2n-1)}{2(2^{len_H}-n)}\right) \cdot \left(1 - \frac{2(q_H+q_{sig}+2n-1)}{\min(q,2^{len_H})}\right).$$

Counting the coin toss at the beginning of the main procedure the above expression should be multiplied with $\frac{1}{2}$.

Let $\epsilon^{(n)}_{Sch}$ be an upper bound for adversary's advantage in a chosen message attack on private key of Schnorr signature scheme, assuming the same adversary's effort is like in the algorithm above. Then:

$$\delta^{(n)}_{eq,usd} + \delta^{(n)}_{eq,unsd} \le 2\epsilon^{(n)}_{Sch} \cdot \left(1 - \frac{(q_H+q_{sig}+2n)\cdot(q_H+q_{sig}+2n-1)}{2(2^{len_H}-n)}\right)^{-1} \cdot \left(1 - \frac{2(q_H+q_{sig}+2n-1)}{\min(q,2^{len_H})}\right)^{-1}.$$

Since the attack on signature's private key is not harder than the attack on DLP defined in the same group of prime order (we have proved by the first of the two algorithms that the case "$\ell_j \neq \ell'_j \bmod q$" is negligibly close to DLP, cf. (3)) the theorem is prover. ∎

3.2 Malicious TSA

3.2.1 Reusing commitments.

Subsect. 3.1 covers the case of an external attacker. Suppose now that TSA is going to issue two timestamps for different H_j, H'_j for the same commitment c_j. So what are the chances of malicious TSA to do it unnoticeably? The following theorem presents the key property of the scheme:

Theorem 3.5 *Assuming the standard model, a TSA re-using a commitment c_j for another timestamp would either find $\log_g h$ or would leak its own private key x or would break non-repudiation of Schnorr signature scheme.*

Sketch Suppose we have two different timestamps for the same commitment c_j, made for different messages $\mathrm{H}_j, \mathrm{H}'_j$ (cf. records (1)). Thereby we have two triples (e_j, s_j, ℓ_j), (e'_j, s'_j, ℓ'_j). If $\ell_j \neq \ell'_j \bmod q$, then $\log_g h$ is found by the TSA (compare lines 11-13 of Algorithm 5). Assume that $\ell_j = \ell'_j \bmod q$. If $e_j \neq e'_j \bmod q$, then the private key x leaks in the two timestamps (compare lines 29 and 32 of Algorithm 6 , in present scenario $\tilde{x} = 0$). On the other hand, if $e_j = e'_j \bmod q$, then non-repudiation of Schnorr signatures is broken for the hash function inputs:

$$(H(\mathrm{HS}_{j-1}), \mathrm{H}_j, c_{2j}, c_{2j+1}, \ell_j, j)||r_j \quad \text{and} \quad (H(\mathrm{HS}_{j-1}), \mathrm{H}'_j, c_{2j}, c_{2j+1}, \ell_j, j)||r_j$$

(note that $\mathrm{H}_j \neq \mathrm{H}'_j$, and due to re-usage of the commitment c_j and due to the assumption $\ell_j = \ell'_j \bmod q$ we get that r_j must be the same in both inputs, which is essential in breaking non-repudiation of Schnorr signatures). ∎

Consequently, if $\log_g h$ is known to nobody and DLP in $\langle g \rangle$ is hard, and if Schnorr signatures are hard to repudiate, then making two timestamps for the same commitment would imply leak of the private exponent x of TSA.

3.2.2 Misbehavior scenarios.

In general, a malicious TSA may try the following tricks:

1. change a timestamp after issuing,
2. fork a chain of timestamps.
3. insert an additional timestamp in the past,
4. remove a timestamp.

Let us discuss each of these cases:

ad 1) this trick is covered by Theorem 3.5 . Under the assumptions above the private key x must leak. Presenting the key x by a third party can be regarded in this case as a proof of breaking the scheme either by cryptanalysis or misbehavior of TSA – in both cases business of TSA should terminate.

ad 2) forking a chain at some place also requires creating two timestamps based on the same commitment, so the same remarks as above apply.

ad 3) inserting a timestamp is impossible since all places in the chain should be taken. Indeed, it is impossible to leave an empty space in the chain, since each timestamp is connected with the previous one and therefore such gaps must already have some fixed contents (even empty). Trying to reuse these places means delivering a new timestamp at a given point (see $H(\mathrm{HS}_{i-1})$ in the record (1)). This leads to the same consequences as in the first case.

ad 4) removing a timestamp is impossible since, among others, the successor of a given timestamp is linked to it via a timestamp signature.

3.2.3 Rebuilding the chain.

Note that any third party that derives x can split the timestamp chain in the following way: assume that the timestamps are given for $t = 1, \ldots, 2m + 1$. Then there are commitments for $t = 2m + 2, \ldots, 4m + 3$ as well, but the party does not attempt to open them. Note that the third party learns all exponents k_i for $i = 1, \ldots, 2m+1$ from the equalities $s_i = k_i - xe_i \bmod q$. Therefore it can redo all steps from $m+1$ through $2m + 1$, thereby issuing own commitments for all k_t for $t = 2m + 2, \ldots, 4m + 3$. Afterwards the third party can arbitrarily extend the chain.

Even if the security breach is evident (since given two chains arising from the same commitment anybody can derive secret key x, and may show the key as an evidence in court of justice), it is not clear which of the chains was the original one. Of course, the above attack requires redoing a large number of timestamps - so a large number of users would witnesses against a new chain. Additionally, this problem can be solved by the following countermeasure:

- from time to time TSA publishes a current timestamp in alternative way (in a newspaper),
- publishing a timestamp is compulsory when the index of the current timestamp is $2m + 1$ and the last published timestamp has index m.

It is easy to see that in this case a forged chain always halts before the next "anchor point" which is the published timestamp.

3.3 Long Term Security

Breaking signature scheme and/or leaking the signing key x of TSA brings some problems. However, as already observed above it means rather making mess than collapsing the current chain of signatures. Nevertheless, it seems to be reasonable to use the private key only for a limited period of time, and then, in order to prevent a future leakage, to destroy the key together with parameters (k, ℓ) used for Pedersen commitments. One can also demand that each timestamp must be confirmed with a digital signature by the requester immediately after receiving the timestamp. In this way one can speed up future disputes.

Note that if in the future the signature key is leaked or derived by some novel cryptanalytic method, this does not mean that an adversary will be able to misuse a Stamp & Extend service. Indeed, the second line of defense are the Pedersen commitments. The adversary will have either to get access to the parameters (k, ℓ) stored for future timestamps by TSA or to break DLP in order to provide arguments (k', ℓ') corresponding to the commitments.

4 Implementation Issues

Let us discuss key issues for scheme usability in practical scenario.

4.0.1 Instant response.

The main advantage of the scheme is that despite of efficient verification, the timestamp is returned immediately. This makes room for application of Stamp&Extend for instance by financial institutions, where the transactions have to be processed without delay.

4.0.2 Verification and communication complexity.

The size of messages from requesters to TSA is constant, on the other hand the size of data necessary for complete verification of a timestamp with a serial number i is proportional to $\lfloor \log i \rfloor$, so in practice it is small enough. Moreover, TSA's computational effort for making the next timestamp is always constant.

4.0.3 Memory usage.

The scheme requires to hold the lists of timestamps and other auxiliary data. The lengths of these lists is bounded by $2i + 1$, where i is the number of timestamps issued so far.

If timestamping is intended to be a business service (and not for instance as a system log for a high speed system), then the number of timestamps and therefore the memory usage is a minor issue concerning contemporary technology and in no way is a bottleneck.

4.0.4 Necessity of audits.

In the classical scenario described by legal systems in many countries, time stamping requires a certified time-stamping devices, fulfilling some security profile. This process (apart from questionable value of such inspection for public) is costly not only due to the direct cost of examination, but also due to business risks concerning issuing certificates. If a security flaw is overlooked (which is quite likely in complicated cases), then the certification body can be sued and the height of compensation might be extreme.

A nice feature of Stamp&Extend system is that honesty of TSA can be forced without an external control. The only involvement of a third party is issuing the certificate $Cert$. However, this is a single certificate and the risk can be reduced by compulsory publishing the certificate in a non-electronic way (newspaper).

The audit of the chain of timestamps is done by the requesters. Of course, each requester makes only a small part of the job, but this makes it reasonable from business point of view: the requester does what is vital for his own interests.

4.0.5 Low-end time stamping.

TSA might implement the service just on a PC and a reasonable (but cheap) device for issuing signatures and for exponentiations needed for commitments. The risk and responsibility for such a service can be defined accordingly. The upside of this solution is that security of the system has black-white nature: in case of a problem the system collapses in an undeniable way.

With many (academic) reservations against such solutions, one has to keep in mind that most systems successful in practice are not overshooting security requirements and work with quite modest security level.

4.0.6 Evidence in case of frauds.

In case of a forgery made by TSA the evidence of fraud is undeniable in cryptographic sense. So there is no threat that a judge will deliver a judgment favorable for TSA even in a case when TSA is evidently guilty. This reduces to minimum the risk regarding lawsuits against malicious TSA.

5 Conclusions

We have presented a new timestamping scheme. Similarly like in [LBG08a] timestamp requests are served instantly. However, different security assumptions lead us to a centralized service with a setup phase reduced to generating a single commitment only. Moreover, security reductions for the protocol presented above are provided.

References

[BdM91] Josh Cohen Benaloh and M. de Mare. Effcient broadcast time-stamping. Technical Report TR-MCS-91-1, Clarkson University Department of Mathematics and Computer Science, August 1991.

[BdM93] Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In Tor Helleseth, editor, *EUROCRYPT*, volume 765 of *LNCS*, pages 274–285. Springer, 1993.

[BG06] Kaouthar Blibech and Alban Gabillon. A new timestamping scheme based on skip lists. In Marina L. Gavrilova, Osvaldo Gervasi, Vipin Kumar, Chih Jeng Kenneth Tan, David Taniar, Antonio Laganà, Youngsong Mun, and Hyunseung Choo, editors, *ICCSA (3)*, volume 3982 of *LNCS*, pages 395–405. Springer, 2006.

[BKK10] Przemyslaw Blaskiewicz, Przemyslaw Kubiak, and Miroslaw Kutylowski. Two-head dragon protocol: Preventing cloning of signature keys - work in progress. In Liqun Chen and Moti Yung, editors, *INTRUST*, volume 6802 of *LNCS*, pages 173–188. Springer, 2010.

[BL06] Ahto Buldas and Sven Laur. Do broken hash functions affect the security of time-stamping schemes? In Jianying Zhou, Moti Yung, and Feng Bao, editors, *ACNS*, volume 3989 of *LNCS*, pages 50–65, 2006.

[BLSW05] Ahto Buldas, Peeter Laud, Märt Saarepera, and Jan Willemson. Universally composable time-stamping schemes with audit. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *ISC*, volume 3650 of *LNCS*, pages 359–373. Springer, 2005.

[BN08] Ahto Buldas and Margus Niitsoo. Can we construct unbounded time-stamping schemes from collision-free hash functions? In Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, editors, *ProvSec*, volume 5324 of *LNCS*, pages 254–267. Springer, 2008.

[BN10] Ahto Buldas and Margus Niitsoo. Optimally tight security proofs for hash-then-publish time-stamping. In Ron Steinfeld and Philip Hawkes, editors, *ACISP*, volume 6168 of *LNCS*, pages 318–335. Springer, 2010.

[BS04] Ahto Buldas and Märt Saarepera. On provably secure time-stamping schemes. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *LNCS*, pages 500–514. Springer, 2004.

[BSI12] BSI. Elliptic Curve Cryptography. Technische Richtlinie TR-03111 v2.0, 28 June 2012.

[DM10] Ivan Damgård and Gert Læssøe Mikkelsen. Efficient, robust and constant-round distributed RSA key generation. In Daniele Micciancio, editor, *TCC*, volume 5978 of *LNCS*, pages 183–200. Springer, 2010.

[Eur00] European Parliament and of the European Council. Directive 1999/93/ec of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures. *Official Journal of the European Communities*, L(13), 19/1/2000.

[Eur12] European Commision. Proposal for a regulation of the European Parliament and of the Council on electronic identification and trust services for electronic transactions in the internal market, 4th June 2012.

[HS91] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *J. Cryptology*, 3(2):99–111, 1991.

[Kal02] Burton S. Kaliski Jr. On hash function firewalls in signature schemes. In Bart Preneel, editor, *CT-RSA*, volume 2271 of *LNCS*, pages 1–16. Springer, 2002.

[KKK12] Łukasz Krzywiecki, Przemysław Kubiak, and Mirosław Kutyłowski. Stamp and extend - instant but undeniable timestamping based on lazy trees. In Chris J. Mitchell and Allan Tomlinson, editors, *INTRUST*, volume 7711 of *Lecture Notes in Computer Science*, pages 5–24. Springer, 2012. available online: `http://dx.doi.org/10.1007/978-3-642-35371-0_2`.

[LBG08a] Duc-Phong Le, Alexis Bonnecaze, and Alban Gabillon. A secure round-based timestamping scheme with absolute timestamps (short paper). In R. Sekar and Arun K. Pujari, editors, *ICISS*, volume 5352 of *LNCS*, pages 116–123. Springer, 2008.

[LBG08b] Duc-Phong Le, Alexis Bonnecaze, and Alban Gabillon. Signtiming scheme based on aggregate signature. In *ISI*, pages 145–149. IEEE, 2008.

[Lip02] Helger Lipmaa. On optimal hash tree traversal for interval time-stamping. In Agnes Hui Chan and Virgil D. Gligor, editors, *ISC*, volume 2433 of *LNCS*, pages 357–371. Springer, 2002.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *LNCS*, pages 129–140. Springer, 1991.

[Sch91a] Claus Schnorr. *Method for identyfying subscribers and for generating and veryfing electronic signatures in a data exchange system.* U.S. Patent 4,995,082, 1991.

[Sch91b] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.