A Secure Round-based Timestamping Scheme with Absolute Timestamps*

Duc-Phong Le, Alban Gabillon, and Alexis Bonnecaze

No Institute Given

Abstract. The aim of timestamping systems is to provide a proof-of-existence of a digital document at a given time. Such systems are important to ensure integrity and non-repudiation of digital data over time. Most of the existing timestamping schemes use the notions of round (a period of time) and round token (a single value aggregating the timestamping requests received during one round). Such schemes have the following drawbacks: (i) Clients who have submitted a timestamping request must wait for the end of the round before receiving their timestamping certificate (ii) TimeStamping Authorities (TSA) based on such schemes are discrete-time systems and provide relative temporal authentication only, i.e. all the documents submitted during the same round are timestamped with the same date and time. (iii) the TSA can tamper timestamps before the round token is published in a widely distributed media. In this paper, we define a new timestamping scheme which overcomes these drawbacks. First, we use chameleon hash functions to compute the round token at the beginning of the round. Therefore, timestamping certificates can be returned to clients immediately after receiving their request. There is no need to wait for the end of the round. Second, our scheme is distributed, i.e. several servers cooperate to compute round tokens and timestamping certificates. Such a distribution of servers allows us to define a continuous time scheme secure against all kind of attacks and providing us with provable absolute temporal authentication.

Keywords: Timestamping scheme, Merkle tree, Chameleon hash function, Absolute timestamp.

1 Introduction

The use of digital documents is growing rapidly, nowadays. It thus becomes very important to ensure their security when they are stored/exchanged on the open network environment. Cryptographic primitives, including digital signatures, help to provide ongoing assurance of authenticity, data integrity, confidentiality and non-repudiation. Besides that, it is also important to be able to certify that an electronic document has been created at a certain date. Timestamping protocols, which prove the existence of a message/document at a certain

* This work was supported by Conseil Général des Landes and the French Ministry for Research under Project ANR-07-SESU-FLUOR.

time, are mandatory in many domains like patent submissions, electronic votes or electronic commerce, where frauds are related to monetary (or even political) interests.

Moreover, timestamping services can serve as non-repudiation services. A digital signature is only legally binding if it was made when the user's certificate was still valid. In reality, to prevent eventual compromise concerning the private key, key pairs used in public key cryptosystems have a limited lifetime which can be shorter than the document time-to-life. Therefore, the digital signature is not sufficient to guarantee non repudiation. In order to repudiate a signature, a malicious signer could claim that his credentials were already compromised when the signature was issued. A secure timestamp bound to the signed document can prevent such repudiation. Indeed, thanks to the timestamp, it becomes possible to determine, at a later time, if the document was signed (using the owner's private signature key) within the validity period of the certificate or not.

Most of the existing timestamping schemes use the notions of round (a period of time or a number of requests) and round token (a single value aggregating the timestamping requests received during one round). Such schemes have the following drawbacks: (i) Clients who have submitted a timestamping request must wait for the end of the round before receiving their timestamping certificate (ii) TimeStamping Authorities (TSA) based on such schemes are discrete-time systems and provide relative temporal authentication only, i.e. all the documents submitted during the same round are timestamped with the same date and time. (iii) the TSA can tamper timestamps before the round token is published in a widely distributed media.

In this paper, we define a new round-based timestamping scheme which overcomes these drawbacks. First, we use chameleon hash functions to construct the authenticated data tree (in this paper, we use the Merkle tree) and to compute the round token at the beginning of the round. Second, our scheme is distributed, i.e. several servers cooperate to compute round tokens and timestamping certificates.

Chameleon hash function was introduced by Krawczyk and Rabin in [17] for the purpose of constructing chameleon signatures. A chameleon hash function is a trapdoor collision resistant hash function. It differs from other conventional one-way hash methods in that there exists a private or trapdoor key associated with the chameleon hash function that makes it computationally feasible for the owner of the private key to find collisions; where finding a collision is defined as the ability to compute multiple messages which map to the same hash value. Formally, let $\mathcal{H}_r : \{0,1\}^m \times \{0,1\}^r \mapsto \{0,1\}^k$ be a chameleon hash function associated with a hashing key HK and a trapdoor key TK. Then, it is easy to find r such that $\mathcal{H}_r(m,r) = \mathcal{H}_r(m_0,r_0)$ when (HK,TK) and (m,m_0,r_0) are given, however, it is hard to find two messages m, m_0 and two auxiliary numbers r, r_0 such that $\mathcal{H}_r(m,r) = \mathcal{H}_r(m_0,r_0)$ when only HK is given.

For a given round, our timestamping procedure consists of two phases. The first phase is performed off-line before the round begins (before receiving the timestamping requests) and the second phase is performed on-line after the beginning of the round (while receiving the requests). Using a chameleon hash function allows the TSA to pre-construct an authenticated data tree and pre-generate timestamps and round token in the off-line phase. Later, in the on-line phase, when the i^{th} request of the round arrives, the TSA only needs to find a collision for the timestamp at the position i in the authenticated data tree and returns the i^{th} timestamp to the client. Consequently, a timestamping certificate with absolute time can be returned to the client immediately after receiving his request. The same technical idea was used to construct on-line/off-line signatures [21].

The biggest danger of using a chameleon hash function lies in the possible compromission of the trapdoor key TK. We eliminate such a risk by distributing the trapdoor key in a network of servers. Each server knows only a fragment of the trapdoor key TK. Therefore, any calculation requiring the trapdoor key can only be done by a collaboration of a given number of servers. This number λ is called the threshold. We need a collaboration of at least λ servers to get a result. Such a system is called a (λ, n)-*threshold cryptographic system*. In our scheme, if k is the number of corrupted servers, then the threshold must be greater than $2k$ in order for the calculation to succeed. The use of a threshold scheme has two objectives:

1. Avoiding the compromission of the trapdoor key.
2. Proving the absolute time. The absolute time must be checked by at least λ servers. In our scheme, we need $\lambda = 2k + 1$, when the number of failed or malicious servers is at most k[1].

Threshold secret sharing protocols frequently require one trusted dealer which generates and gives the secret to the n servers. In order to eliminate such a trusted dealer, we use a Distributed Key Generation (DKG) protocol to generate the hashing key HK and trapdoor key TK.

The remainder of this paper is organized as follows. Section 2 introduces some basic concepts about timestamping. Section 3 previews some basic notations used in this paper, chameleon hash functions and security requirements for timestamping schemes. In Section 4, we present our scheme. In Section 5, we analyze its security. Finally, we conclude the paper in Section 6.

2 Basic Concepts

2.1 Naive Protocol

The first idea on timestamping is naive. Whenever a client has a document to timestamp, he or she transmits the document D to the timestamping service (TSS); the service timestamps D and retains a copy of the document for safe-keeping and verifying. Such protocol has a lot of problems: privacy, bandwidth, storage and incompetence. Solutions using cryptographic tools like secure collision-resistant hash functions and digital signatures were proposed to improve

[1] We require $k < n/3$ to guarantee robustness

the naive solution [14]. However, these solutions require a trusted service (with a precise clock) that provides data items with current time value and digitally signs them [1]. The assumption of unconditionally trusted service hides a risk of possible collusions that may not be acceptable in applications. The risks are especially high in centralized applications. The first attempt to eliminate trusted services from timestamping schemes was made by Haber and Stornetta in [14]. They proposed two rather different approaches: linking scheme and distributed trust scheme. The first one relies on a centralized server model that has to be trusted. The idea behind this scheme is to prevent the server from forging fake time-stamp tokens by linking linearly the timestamps in a chronological chain. The second approach consists in distributing the required trust among the users of the service.

2.2 Absolute vs. Relative Temporal Authentication

The main objective of timestamping schemes is temporal authentication. Existing timestamping schemes provide two types of temporal authentication: *absolute authentication* [1, 14] and *relative authentication* [2, 10, 8, 4]. Absolute authentication provides absolute timestamps positioning the document at a particular point in time, based upon the time given by a trusted, mutually agreed upon source. Existing absolute authentication schemes presuppose that the TSS is a trusted entity. Relative authentication provides relative timestamp containing information that only allows verifying if a document was timestamped before or after another document. For the relative scheme the existence of a trusted entity is not necessary. There are mechanisms which guarantee that a document will always be timestamped with the current date and hour even if the TSS is malicious. For applications like patent submissions, electronic votes, ticket bookings ... the relative authentication is enough to be applicable. However, the absolute timestamp is very important for some applications (e.g. for contracts, bills ...). In this case, we need to know exactly when a contract (or a bill) was signed or created. Besides, the absolute timestamp also allows us to compare timestamps generated by several TSAs that use the same or a different scheme.

2.3 Existing timestamping schemes

Simple schemes : A simple timestamping scheme is typically as the above naive solution. Such a scheme can use cryptographic tools like hash functions, digital signatures to guarantee the confidentiality and the integrity of documents. The timestamp tokens are independent from each other. Simple schemes are straightforward and easy to implement. On the other hand, their main weakness is that the TSA has to be trusted unconditionally. Since the TSA is the entity that guarantees the correctness of the time parameter, a malicious TSA can backdate or forward-date timestamps.

Linking schemes : The first linking scheme which links linearly timestamps was proposed by Haber and Stornetta [14]. The linear linking scheme poses a very high demand on cooperation among clients, the verification cost thus is very expensive. Moreover, in order to prevent *fake sub-chain* attacks which were showed in [16], it may impose a long computation time before a trusted timestamp is encountered on the chain, it is thus *impractical.*

Later, various improvements for the linking scheme have been proposed in [3, 2, 15, 10, 8, 4]. These schemes use the notion of round: a round can be a given number of requests, a period of time or a combination of both. At the end of each round a round token is calculated which depends on all the requests submitted during that round and on the previous round token. This allows the TSA to reduce the amount of information to be stored (for verification) and to improve system scalability. Subsequently, round tokens are regularly published in a widely distributed media (e.g. a newspaper). After the publication it becomes impossible to forge timestamps (either to issue fake ones afterward, or modify already issued ones), even for the TSA. Some of these schemes are said to be partially ordered [3, 2, 15] while others are said to be totally ordered [8, 4].

– With partially ordered schemes, documents submitted during a given round are all timestamped with the same time.
– With totally ordered schemes, documents submitted during a given round are still all timestamped with the same time, but it becomes possible to order the requests and answer the question "was this document submitted before that one ?".

The main drawbacks of existing round-based schemes are the followings:

– The time attached to a document is the time corresponding to the round. Consequently, all the documents which were submitted during the same round were timestamped with the same time. It is important to note that even totally ordered schemes are not continuous time schemes and comparing timestamps issued from different totally ordered TSS can be impossible.
– Timestamps are generated and sent to clients at the end of the round. Thus, the TSA can alter timestamps if the round is not yet finished. It can even tamper timestamps as long as one reference token has not been published in a widely distributed media.

Distributed timestamping schemes : In such a scheme, trust is needed among the users of the service [14] or among the network of trusted servers [6, 23, 7]. In Haber-Stornetta's scheme [14], Alice who would like to timestamp a document sends her request to a set of k users of the service and receives in return from these users a signed message that includes the time t. Her timestamp consists of k signatures. The main drawback of this protocol is the number of available users needed to answer to Alice's request.

A distributed approach based on a network of TSAs is proposed by Bonnecaze et al. [6, 7] and Tulone [23]. In general, these schemes use a threshold scheme by fault-tolerantly distributing the secret information (e.g. the key which is used to sign the document) among a cluster of trusted servers. A subset of the network of TSAs signs and attaches the time t to the document when a new request arrives. In order to make backdating possible, all involved TSAs have to become part of the malicious attack. Distributed schemes thus decrease the dependence on the TSA and also increase the availability of the timestamping service and resistance to Denial of Service attacks. Even though it is secure, the high number of interactions between servers makes the scheme difficult to implement. Furthermore, like in simple timestamping schemes, distributed schemes should store all timestamping certificates for verifying afterward.

Another drawback of both above approaches is the lifetime of the signatures which were sent to clients by trusted servers (or by the users of service). If the keys pair of a signer is expired, valid timestamps cannot be verified.

3 Preliminary

3.1 Notations

We denote by $\{0,1\}^*$ the set of all (binary) strings of finite length. If X is a string then $|X|$ denotes its length in bits. If X, Y are strings then $X\|Y$ denotes an encoding from which X and Y are uniquely recoverable. If S is a set then $X \in_R S$ denotes that X is selected uniformly at random from S. For convenience, for any $k \in N$ we write $X_1, \ldots, X_k \in_R S$ as shorthand for $X_1 \in_R S, \ldots, X_k \in_R S$.

3.2 Cryptographic Tools

Authenticated Data Structure Authenticated data structures provide cryptographic proofs that their answers are as accurate as the author intended, even if the data structure is being maintained by a remote host. In this section we briefly recall the most known authenticated data structure proposed by Merkle [19], called *Merkle tree*.

A *Merkle tree* is a binary tree with an assignment of a string to each node: $n \mapsto P(n)$, such that the parent's node values are one-way functions of the children's node values.

$$P(n_{parent}) = \mathcal{H}(P(n_{left})\|P(n_{right})),$$

where $\mathcal{H}$ denotes the one-way function, a possible choice of such a function is SHA-1 in practical.

The *authentication path* of a leaf $leaf_i$, denoted $auth_i$, consists of the interior nodes that are siblings on the path from the root to the leaf $leaf_i$.

Chameleon Hash Function The principal cryptographic tool we use in this paper is a *chameleon hash function*. Informally, a chameleon hash function is a special type of hash function, whose collision resistance depends on the user's state of knowledge: Without knowledge of the associated trapdoor, the chameleon hash function is resistant to the computation of pre-images and of collisions. However, with knowledge of the trapdoor, collisions are efficiently computable.

Definition 1. *A chameleon hash function [17], also called trapdoor hash function, is associated with a public (hashing) key, denoted HK, and a corresponding private key (the trapdoor for finding collisions), denoted TK. The chameleon hash function, denoted $\mathcal{H}_r$ can be computed efficiently given the value of HK. On input a message $m \in \mathcal{M}$ and a random string $r \in_R \mathcal{R}$, the function generates a hash value $\mathcal{H}_r(m, r)$ which satisfies the following properties :*

- ***Collision Resistance***. *There is no probabilistic polynomial time algorithm $\mathcal{A}$ that on input the public key HK outputs, with a probability which is not negligible, two pairs (m_1, r_1) and (m_2, r_2) such that $\mathcal{H}_r(m_1, r_1) = \mathcal{H}_r(m_2, r_2)$.*
- ***Trapdoor Collisions***. *There exists a polynomial time algorithm $\mathcal{A}$ such that on inputs the pair (HK, TK), a pair (m_1, r_1), and a message m_2, then $\mathcal{A}$ outputs r_2 such that :*
  - $\mathcal{H}_r(m_1, r_1) = \mathcal{H}_r(m_2, r_2)$.
  - *If r_1 is distributed uniformly, m_1, and (m_2, r_2) such that $H(m_1, r_1) = H(m_2, r_2)$, then r_2 is computationally indistinguishable from uniform in $\mathcal{R}$.*

Chameleon hash functions can be constructed from the hardness of factoring, discrete logarithm problem. Reader can refer to [17] for more details.

Shamir Secret Sharing Shamir's secret sharing scheme [20] is a threshold scheme based on polynomial interpolation. It allows a dealer D to distribute a secret value s to n players, such that at least $\lambda < n$ players are required to reconstruct the secret. In this section, we briefly recall his protocol.

To share the secret s among players $P_1, P_2, \ldots, P_n$, such that λ players are required to reconstruct the secret:

1. Dealer D creates a random polynomial $f(x)$ of degree $\lambda - 1$:
$$f(x) = a_0 + a_1 x + \cdots + a_{\lambda-1} x^{\lambda-1}.$$
This polynomial is constructed over a finite field, such that the coefficient a_0 is the secret s and all other coefficients are random elements in the field, the field is known to all participants.
2. Dealer D publicly chooses n random distinct evaluation points: $x_j \neq 0$, and secretly distributes to each player P_j the share $s_j = f(x_j), j = 1 \ldots n$.

To reconstruct the secret from shares s_1, s_2, ..., s_k:

1. Use Lagrange interpolation to find the unique polynomial $f(x)$ such that $\deg f(x) < \lambda$ and $f(x_j) = s_j$ for $j = 1, 2, \ldots, k$.
2. Reconstruct the secret to be $f(0)$.

Lagrange interpolation:
$f(x) = \sum_{i=1}^{\lambda} f(i) l_i(x)$, where $l_i(x)$ is the Lagrange polynomial: $l_i(x) = \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)}$ which has value 1 at x_i, and 0 at every other x_j.

3.3 Security Requirements

The security objectives of timestamping schemes is to guarantee the properties: *correctness*, *data integrity* and *availability*. The first property requires that the verification succeeds only if T_x is correct. The second means that an adversary cannot tamper the timestamp by back or forward dating it, or modify the request associated to it, or insert an old timestamp in the list of timestamps previously issued. The last property means that timestamping and verification must be available despite processes failures.

In general, there exist two major types of attacks on timestamping protocols: *back-dating* and *forward-dating* attacks. In the former attack, an adversary may try to "back-date" the valid time-stamp. This is a fatal attack for applications in which the priority is based on descendant time order (e.g. patents ...). The adversary may corrupt the TSA and may try to create a forged but valid timestamp token. In the later, an adversary may try to "forward-date" the timestamp without the approval of the valid requester. This is a fatal attack for applications in which the priority is based on ascendant time order (e.g. Will ...).

The forward-dating attacks can be prevented by requiring the client's identity which allows us to determine who had timestamped the document. This type of attacks was analyzed in more details by Matsuo and Oguro in [18]. Thus, security problems of timestamping systems only concentrate on back-dating attacks. The simple protocol is clearly not secure against back-dating attacks if TSA is corrupted, but the linking protocol is since a verifier can check the validity by computing the chain of hash values using published hash values. The strongest security condition against back-dating attacks for a timestamping scheme is presented by Buldas and Laur [9] which is defined using the following attack-scenario with a malicious Server:

1. TSA computes and publishes a commitment R. Note that TSA is assumed to be *malicious*, so there are no guarantees that R is created by aggregating a set X of requests.
2. Alice, an inventor (client), creates a description $D_A \in \{0,1\}^*$ of her invention and protects it somehow, possibly by filing a patent or obtaining a time stamp.
3. Some time later, the invention D_A is disclosed to the public and the TSA tries to steal it by showing that it knew the invention long before Alice timestamped it. The TSA creates a slightly modified version D'_A of D_A (at least the author's name should be replaced).
4. Finally, the TSA back-dates a hash value $x' = H(D'_A)$ of the modified invention document, by finding a timestamping certificate $C_{x'}$, so that $\mathsf{Verify}(x', C_{x'}, R) = 1$. The TSA can then use this timestamp to claim his rights to the invention.

4 Our Construction

In this section, we describe a round-based distributed timestamping scheme which provides absolute temporal authentication. The basic idea is to make use of the chameleon hash function to pre-construct a Merkle tree before each round. When the TSA receives the i^{th} timestamping request, it finds a collision at the i^{th} position in the Merkle tree and immediately returns a timestamping certificate to the client. In order to guarantee the secret of the trapdoor key of the chameleon hash function, our scheme uses a threshold secret sharing scheme which enables the trapdoor key to be shared among n servers such that a subset of them can find a trapdoor collision of the chameleon hash function without reconstructing the key. In other words, at most k of the n servers in a threshold timestamping scheme may be compromised without endangering the security of the timestamping scheme. Our timestamping scheme is robust with $k < n/3$ and can detect compromised servers when they try to generate a faulty partial trapdoor collision.

The timestamping service consists of two types of servers: a reception server and several hash servers. The former will receive requests, add time and return timestamps to clients. The later consists of n servers, each of them keeps a fragment of the trapdoor key of the TSA and will make a partial trapdoor collision of the chameleon hash function when a new timestamping request will arrive. The trapdoor key TK is generated and stored in distributed manner. This key can be regularly renewed after some rounds or when a number of corrupted servers is superior to a threshold k. Before each round, (m_i', r_i') pairs are also generated and stored in a distributed manner in the hash servers. Then, the reception server generates hash digests $h_i = \mathcal{H}_r(m_i', r_i')$, constructs the Merkle tree and publishes the round token and the hashing key HK in a newspaper. When Alice sends a document she want to have timestamped to the TSA, a subset of $2k+1$ hash servers finds a partial trapdoor collision r_i and then returns a timestamp certificate to the client. In particular, the timestamping scheme works as follows:

Let $\mathcal{P}$ be the set of n hash servers, l be a security parameter of system, r be a security parameter of random strings in the chameleon hash function, $\mathcal{H} : \{0,1\}^* \mapsto \{0,1\}^l$ be a collision-resistant hash function (e.g. SHA-1) and $\mathcal{H}_r : \{0,\ 1\}^l \times \{0,\ 1\}^r \mapsto \{0,1\}^l$ be a chameleon hash function. In describing, we use the Merkle tree and the log-discrete-based chameleon hash function $\mathcal{H}_r(m,r) = \mathcal{H}_r(g^{r+xm}) = \mathcal{H}_r(g^r h^m)$, where $h = g^x$, x is trapdoor key and g, h are hash keys. This chameleon hash function was presented in [12], and its security relies on *one-more-discrete-logarithm* assumption. We also denote $\mathcal{M} = \{0,1\}^l$ and $\mathcal{R} = \{0,1\}^r$.

Key generation. In describing of the key generation, we make use of the Distributed Key Generation (DKG) protocol for discrete-log-based threshold cryptosystems of Gennaro et al. [13] to securely generate keys and m_i', r_i' pairs.

1. Use the DKG protocol to create $h = g^x$, where $x \in_R \mathbb{Z}_p$ is the trapdoor key and $P_j \in \mathcal{P}, 1 \leq j \leq n$ receives the share x_j for a degree k polynomial $p_x(y) \in \mathbb{Z}_p[y]$ such that $p_x(0) = x$.
2. Publish the hashing keys g, h. Each hash server $P_j \in \mathcal{P}$ retain x_j.

Setup. Suppose that 2^m be the number of requests pre-generated for each round. For $1 \leq i \leq 2^m$:

1. Use the DKG protocol to create $g^{r'_i}$, where $r'_i \in_R \mathcal{R}$. Each hash server $P_j \in \mathcal{P}$ receives the share r'_{ij} for a another degree k polynomial $p_{r'_i}(y) \in \mathbb{Z}_p[y]$ such that $p_{r'_i}(0) = r'_i$.
2. Use the DKG protocol to create $h^{m'_i}$, where $m'_i \in_R \mathcal{M}$. Each hash server $P_j \in \mathcal{P}$ receives the share m'_{ij} for a another degree k polynomial $p_{m'_i}(y) \in \mathbb{Z}_p[y]$ such that $p_{m'_i}(0) = m'_i$.
3. Use the DKG protocol to generate shares z_{ij} for each hash server $P_j \in \mathcal{P}$ of a degree $2k$ polynomial $p_0(y) \in \mathbb{Z}_p[y]$ such that $p_0(0) = 0$.
4. Now $g^{r'_i}$ and $h^{m'_i}$ are both known to the servers, so the hash digest $h_i = \mathcal{H}_r(m'_i, r'_i) = \mathcal{H}_r(g^{r'_i} h^{m'_i})$ can be computed. The reception server computes h_i for $i = 1, \ldots, 2^m$.
5. The reception server constructs the Merkle tree whose leaves are above hash digests and computes tree root (round token). Then the TSA publishes the round token, the hashing key (g, h) of the chameleon hash function $\mathcal{H}_r$.
6. Generate a sequence of authentication paths $auth_i$, one for each leaf. These paths are stored by the reception server.

To compute parent's node value in constructing the Merkle tree from hash digests h_i, we use a cryptographic hash function $\mathcal{H}$ (e.g. SHA-1).

Timestamping. The Stamping Protocol used to generate a timestamp works as follows:

1. Alice, the client i^{th} of the round sends her identity and the hash value of the document D_i she wants to have timestamped: she sends $ID_A, m_i\ (= \mathcal{H}(D_i))$.
2. The reception server adds the current time t to m_i, computes $m = \mathcal{H}(m_i \| t)$ and then sends t, m_i, m to all of the servers.
3. Each hash server $P_j \in \mathcal{P}$ checks the correct time t (within reasonable limits of precision) and checks whether $m = \mathcal{H}(m_i \| t)$. If so, P_j computes $c_{1j} = r'_{ij} - x_j m$ and $c_{2j} = x_j m'_{ij} + z_{ij}$ which is P_j's share of the trapdoor collision. The role of the share z_{ij} in c_{2j} is to allow us make the polynomial random. Then P_j sends c_{1j}, c_{2j} to the reception server and to all of the other server in P. After receiving $2k + 1$ shares[2] from a subset $\mathcal{P}'$ of hash servers $\mathcal{P}$, the reception server:

[2] Our protocol requires a multiplication operation of two secrets x and m, so each share will be a degree $2k$ polynomial. For this reason, we need $2k+1$ servers for each calculation

4. Defines $f_j(y) = \prod_{P_l \in \mathcal{P}' \setminus P_j} \frac{l-y}{l-j}$, as in the definition of Lagrange interpolation. The trapdoor collision is computed as follows:
$$\begin{aligned} r_i &= \sum_{P_j \in \mathcal{P}'} (c_{1j} + c_{2j}) f_j(0) \\ &= \sum_{P_j \in \mathcal{P}'} (r'_{ij} - x_j m + x_j m'_{ij} + z_{ij}) f_j(0) \\ &= r'_i + x m'_i - x m. \end{aligned}$$
5. Each hash server $P_j \in \mathcal{P}'$ discards its share m'_{ij}, r'_{ij}, z_{ij}.
6. The TSA then returns the timestamp certificate $C_i = (i, ID_A, m_i, r_i, t, auth_i, t_r)$ to Alice, where i is the certificate serial number, and t the current date and time, t_r is the date and time for the round and $auth_i$ is the authentication path of the message m_i on the Merkle tree (for the purpose of reconstructing this timestamp). This is the timestamp for Alice's document D_i.
7. Alice receives the certificate and checks that it contains the hash of the document she asked a timestamp for and the correct time (within reasonable limits of precision).

Verification. A verifier who questions the validity of the timestamp C_i for the document D_i will:

1. Check that the hash value $m_i = \mathcal{H}(D_i)$ corresponds to the document D_i.
2. Compute the value of the leaf i: $h_i = \mathcal{H}_r(\mathcal{H}(m_i \| t), r_i)$ check that it is part of the data that reconstructs the timestamp for the round.

Furthermore, the scheme is robust against dishonest hash servers. As pointed out in [12], we can verify values c_{1j}, c_{2j} for the purpose of detecting incorrect shares using zero-knowledge proofs for verification. In this section, we briefly recall the proof of Crutchfield et al. in [12].

1. Verifying c_{1j}. Because $g^{r'_{ij}}$ and g^{x_j} are known values from the DKG protocol, we can compute for each hash server $P_i \in \mathcal{P}'$, $g^{r'_{ij}} \cdot (g^{x_j})^{-m} = g^{r'_{ij} - x_j m}$ and confirm that $g^{c_{1j}} = g^{r'_{ij} - x_j m}$ as desired.
2. Verifying c_{1j}. We can apply Chaum and Pedersen's zero knowledge proofs (ZKP) for equality of discrete logarithms [11]. Let $d = g^{x_j}, e = g^{m'_{ij}}$ and $f = g^{c_{1j} - z_{ij}}$. Each hash server P_j uniformly chooses $r \in Z_p$ at random and computes $H(g, d, e, f, g^r, e^r) = c$, where H is a random oracle and c is the challenge. P_j computes $v = x_j c + r$ and broadcasts the pair (c, v). Finally, all servers compute and confirm that $H(g, d, e, f, g^v d^{-c}, e^v f^{-c}) = c$.

If any of the shares is deemed incorrect, then broadcast a complaint against P_j. If there are at least $k + 1$ complaints, then clearly P_i must be corrupt since with at most k malicious players, there can be at most k false complaints.

5 Discussion

5.1 Main features

Our scheme is a secure robust (k, n)-threshold timestamping scheme. Unlike the timestamping scheme in [22] and timestamping schemes based on threshold signature in [5, 23, 7], our scheme is basically based on rounds and the round tokens are regularly published in a widely distributed media. For verifying timestamps, clients only need the trapdoor keys g, h, that are published along with the round token. Thus, our scheme does not face security problems related to the limited lifetime of digital signatures (even threshold signatures). In our scheme, we renew the hash keys g, h and trapdoor key x after several rounds or when a number of corrupted servers is superior to a threshold k.

Like in other round-based linking timestamping schemes, a verifier does not need to communicate with the TSA. Only the knowledge of the published round token is needed to verify a timestamping certificate.

In our scheme, a timestamping certificate is returned immediately to the client after her or his request. Thus, unlike in other Merkle tree-based schemes, our scheme provides the clients with absolute timestamps, i.e. each document is timestamped with its own date and time regardless of the round period. This allows us to know exactly when the document was timestamped and to compare timestamps generated by several TSAs that use the same or a different scheme. In addition, our scheme allows verifiers to verify timestamps before the end of the round. The client who keeps the timestamping certificate of her or his document m cannot change the time t in the timestamping certificate unless she or he finds a collision of the hash digest $m\|t$.

In our scheme, the number of requests in one round is pre-determined as an exponent m of 2 before the round begins. Thus, m should be elaborately chosen. If we choose n large, then the duration of a round can be too long. On the other hand, if m is small, the duration of a round can be too short. Besides, a round can combine between a number of requests 2^m and a period of time. The TSA chooses m large enough. After a period of time (e.g. a week) the TSA discards the rest of pre-generated certificates and terminates the round.

From the point-of-view of efficiency, in our scheme the costs of our key generation and of our construction of Merkle tree are dominated by the cost of the DKG protocol [13]. Our timestamping phase only requires one round of communication between servers. Finally, our verification phase is efficient, it can be executed in an off-line manner by a verifier, i.e. there needs no interaction with the timestamping service.

5.2 Security Analysis

The security of our scheme reduces directly to the security of chameleon hash functions, the security of the Merkle tree (that depends on the property "collision-resistant" of hash functions) and the security of the threshold scheme. Before analyzing the security of our scheme, we consider the following theorem:

Theorem 1 (Collision resistance). *A computationally bound adversary cannot construct two authentication paths $auth_i$ and $auth'_i$ verifiable against the same commitment R if the hash functions used $\mathcal{H}$ and $\mathcal{H}_r$ are collision-resistant.*

Proof. Suppose that an adversary can, in fact, construct an efficient proof collision with proofs $auth_i$ and $auth'_i$ against common commitment R at the same position i. That is, the adversary may find either a trapdoor collision of the chameleon hash function $\mathcal{H}_r$ without knowledge of the trapdoor key or a collision of the hash function $\mathcal{H}$ for the authentication path of leaf i on the Merkle tree. Both of them are impossible if the chameleon hash function $\mathcal{H}_r$ and the hash function $\mathcal{H}$ used for constructing the Merkle tree are collision resistant.

Correctness of the Scheme It is quite easy to see that this property is achieved. From theorem 1, we see that the verification procedure holds if and only if the timestamping certificate is true.

Availability Our scheme is high available. For timestamping, our scheme tolerates the participation of at most $k < \frac{n}{3}$ corrupted servers and requires $2k+1$ servers to construct a timestamping certificate[3]. When the number of corrupted servers reaches k, our scheme allows to change the pair of keys without influencing the correctness of previous timestamping certificates. One can always verify the correctness of a timestamping certificate even if the timestamping service is not available.

Back and Forward Dating Attacks The following security proof of our timestamping scheme against back-dating and forward-dating attacks follows directly from the security of the theorem 1. It is demonstrated in the following theorem:

Theorem 2. *The proposed timestamping scheme is secure against back-dating and forward-dating if the theorem 1 holds.*

Back-dating Consider the following situation, Alice, an inventor, needs to timestamp her patent d at time t. After some time, the invention is disclosed to the public. Bob, an adversary tries to steal the right to Alice's invention. He slightly modifies Alice's invention (at least the author's name should be replaced), and tries to back-date it relative to Alice's invention. Bob is successful if he can construct a timestamp of the modified invention d' such that it can be verified at time t' ($t' < t$). Bob can then use his timestamp to claim his rights on the invention.

Assume that an adversary (who can collude with a subset of servers or not) tries to back-date a document (invention) using a published round token and/or

[3] We require $2k+1$ servers for one calculation and tolerate k corrupted servers, thus k should be inferior $n/3$

some valid timestamps he received at time t' in the past. We distinguish the case where Alice discloses her invention when the round related to the timestamped invention has not yet terminated from the case where the round is over. In the second case, like in other round-based timestamping schemes, the adversary cannot back-date the invention unless he finds a collision for the authentication path related to the invention. In the first case, to back-date the invention, the adversary should collude with at least $2k + 1$ servers. This is impossible since the system only works with at most k malicious servers.

Forward-dating Consider a client who has sign-timestamped the hash of his will which initially favours the adversary. After some time the will is updated, writing the adversary out and then is sign-timestamped again. Assuming that the adversary has access to the hash of the original will, he can once again re-register the hash of the original will. The TSA timestamps the hash of the "first" will and then sends to the adversary a token which can be used to prove authenticity of the "first" will.

As we presented in the section 3.3, these attacks can be prevented by using the client's identity (see a more details discussion in [18]).

6 Conclusion

Our new round-based timestamping scheme delivers certificates instantaneously. This means that clients don't have to wait until the end of the round. The benefit is particularly notable when the length of time of the round is long. Another interesting point is that each certificate provides a provable absolute time. Until now, schemes based on authenticated data structure like Merkle trees did not have this capability. In order to achieve these properties, we used a chameleon hash function and made the scheme distributed. The chameleon hash function allows us to provide an absolute time while the topology of the scheme is important to obtain the desired security. This scheme can be easily implemented since the cryptographic bricks are already well known and partially implemented.

References

1. C. Adams, P. Cain, D. Pinkas, and R. Zuccherato. Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP), 2001.
2. D. Bayer, S. Haber, and W. S. Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences II: Methods in Communication, Security, and Computer Science*, pages 329–334, London, UK, 1993. Springer-Verlag.
3. J. Benaloh and M. de Mare. Efficient broadcast time-stamping. Technical Report 1 TR-MCS-91-1, Clarkson University Department of Mathematics and Computer Science, August 1991.
4. K. Blibech and A. Gabillon. A new timestamping scheme based on skip lists. In *ICCSA (3)*, pages 395–405, 2006.

5. A. Bonnecaze. A multi-signature for time stamping scheme. In *SAR/SSI 06: The 1st Conference On Security in Network Architectures and Information Systems*, Seignosse, France, June 2006.
6. A. Bonnecaze, P. Liardet, A. Gabillon, and K. Blibech. Secure time-stamping schemes: A distributed point of view. *Annals of Telecommunications*, 61(5-6):662–681, May-June 2006.
7. A. Bonnecaze and P. Trebuchet. Threshold signature for distributed time stamping scheme. *Annals of telecommunications*, 62(11-12):1353–1363, 2007.
8. A. Buldas, P. Laud, H. Lipmaa, and J. Villemson. Time-stamping with binary linking schemes. In *CRYPTO*, pages 486–501, 1998.
9. A. Buldas and S. Laur. Do broken hash functions affect the security of time-stamping schemes? In *ACNS*, pages 50–65, 2006.
10. A. Buldas and H. Lipmaa. Digital signatures, timestamping and the corresponding infrastructure. Technical report, Kberneetika AS, 2000.
11. D. Chaum and T. P. Pedersen. Wallet Databases with Observers. In Ernest F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 1992.
12. C. Crutchfield, D. Molnar, D. Turner, and D. Wagner. Generic on-line/off-line threshold signatures. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 58–74. Springer, 2006.
13. R. Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptol.*, 20(1):51–83, 2007.
14. S. Haber and W. S. Stornetta. How to Time-Stamp a Digital Document. In *CRYPTO '90: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, pages 437–455, London, UK, 1991. Springer-Verlag.
15. S. Haber and W. S. Stornetta. Secure names for bit-strings. In *ACM Conference on Computer and Communications Security*, pages 28–35, 1997.
16. M. Just. Some timestamping protocol failures. In *NDSS 98: Proceedings of the Symposium on Network and Distributed Security*, pages 89–96, San Diego, CA, USA, March 1998.
17. H. Krawczyk and T. Rabin. Chameleon signatures. In *NDSS*, 2000.
18. S. Matsuo and H. Oguro. User-side forward-dating attack on timestamping protocol. In *Proc. of the 3rd International Workshop for Applied Public Key Infrastructure (IWAP'04)*, pages 72–83, 2004.
19. R. C. Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, 1979.
20. A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
21. A. Shamir and Y. Tauman. Improved online/offline signature schemes. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2001.
22. A. Takura, S. Ono, and S. Naito. A secure and trusted time stamping authority. In *IWS 99: Internet Workshop*, pages 88–93, Osaka, Japan, 1999. IEEE Computer Society.
23. D. Tulone. A secure and scalable digital time-stamping service, 2006.