A Distributed Time Stamping Scheme

A. Bonnecaze
LIF and EURECOM
Université de Provence
France
alexis.bonnecaze@esil.univ-mrs.fr

P. Liardet
LATP
Université de Provence
France
liardet@cmi.univ-mrs.fr

A. Gabillon
LIUPPA/CSYSEC
IUT de Mont de Marsan
France
alban.gabillon@univ-pau.fr

K. Blibech
LIUPPA/CSySEC
IUT de Mont de Marsan
France
k.blibech@etud.univ-pau.fr

Abstract

Time stamping is a technique used to prove the existence of a digital document prior to a specific point in time. In this paper, we define a trusted reliable distributed time stamping scheme. This scheme is based on a network of servers managed by administratively independent entities.

This work was supported by funding from the French ministry for research under 'ACI Sécurité Informatique 2003-2006. Projet CHRONOS (http://acisi.loria.fr/)'.
Kaouthar Blibech holds a Ph.D scolarship granted by the Conseil Général des Landes.

1. Introduction

The advent of electronic commerce have made the security of communication a major concern. Many governments have chosen to communicate and conduct transactions with citizens, businesses, and other agencies in a secure online environment. The security requirements for electronic document exchange are to ensure the integrity of official communications, to protect constituent privacy, to authenticate people and processes and possibly, to control sensitive information. Digital signatures help to provide ongoing assurance of authenticity, data integrity, confidentiality and non-repudiation.

Time-stamping techniques allow us to certify that an electronic document was created at a certain date. This certification is mandatory for a lot of applications in various domains like patent submissions, intellectual property or electronic commerce.

The first time-stamping protocol was presented during Crypto '90 by Haber and Stornetta. One year later, Benaloh and de Mare proposed a formal definition for a time-stamping system based on a set of participants and three protocols [3]. Since then, a lot of new schemes were proposed and their security analysed [11],[5],[6],[12],[8].

Most of them use the concept of trusted Time-Stamping Authority (TSA) which is supposed to be able to securely time-stamp an electronic document.

However, it may be difficult to build a third party server that can be trusted. Indeed a server may be corrupted or victim of denial of service attacks. In fact, we claim that time-stamping schemes relying on a unique third party server, cannot be trusted. Therefore our objective in this paper is to propose a time-stamping scheme using a multiserver architecture.

Our protocol can be shortly described as follows: The protocol uses n third party servers. For each time-stamping request, k servers among the n servers are randomly chosen to process the request. These k servers are said to be the active servers.

The security of our protocol depends on the number n of servers and on the number k of active servers.

The paper is organized as follows. Section 2 briefly analyses the weaknesses of existing protocols. In Section 3, we give the required properties of our model. In Section 4, we analyse the "k among n" scheme: among the n servers of the system, only k are chosen at random to handle a given time-stamping request. Our time-stamping scheme is presented in section 5 and finally we propose a new random generator to obtain k servers among n.

2. Existing protocols and their weaknesses

Most of the existing systems rely on a centralized server model that has to be trusted. For making the server trustworthy and preventing it from forging fake time-stamping tokens, the method generally used is to link the tokens in a chronological chain (see for example [7]). Periodically, a token is published on an unalterable and widely witnessed media like a newspaper. This scheme offers the following advantages:

- The publication provides us with an absolute time.
- After a token has been published at time t, the server cannot forge a fake time-stamping token former to time t.
- Since tokens are linked in a chronological chain, we can chronologically order the requests submitted between two publications.

However, this scheme has the following drawbacks:

- The publication step is costly and not convenient.
- Before the next publication, the server can tamper the tokens which have been issued since the last publication.
- The entire chronological chain must be stored for verification. In order to reduce the amount of information to be stored, most of the protocols use a binary tree structure also called Merkle Tree [13]. This method allows us to reduce to a logarithmic factor the amount of information to be stored. However, protocols using linking informations are not always accurate and efficient. This is trivially the case when the number of time-stamped documents is very small while the frequency of publication is very low (typically a week). In that case, the accuracy of the time-stamp may not satisfy the client. Notice also that a scheme using a binary tree is not efficient when the number of documents is not close to a power of 2.
- Finally, centralized systems are very vulnerable to Denial of Service attacks.

3. Design requirements

Our aim is to design a multi-server time-stamping system which has to meet the following requirements:

1. being independent from any administrative entity (like a country, a multinational company,...);
2. being resistant against a Distributed Denial of Service (DDoS);
3. being resistant against material failures;
4. being robust against an attack involving less than $n/3$ servers. It is known that any protocol can be made provably secure (without any cryptographic assumptions) if and only if less than one third of the involved parties are corrupted;
5. being able to work without ever trusting a particular componant of the system;
6. being able to deliver an absolute time with an *a priori* fixed error of interval time Δt;
7. being able to prove the datation, from the knowledge of the only time-stamp;
8. having a robust, simple and efficient verification protocol.

4. k among n scheme

In this section, we discuss the security of a general scheme lying on a distributed network of n servers where only k servers are involved in the calculation of a particular time-stamp. In the next section, we present our distributed scheme which does not have the security flaws of the general scheme presented in this section.
The k servers are the **active** servers. They are randomly chosen. The two values n and k depend on the required security level.
The **complete time-stamp**, used to verify and to prove the datation, is built from the k time-stamping **fragments** delivered by the active servers. The number of active servers is defined in order to maintain the required security level as well as to minimize the traffic inside the network.
The model has n servers. Among them, f are supposed to be failed (FS). Among these f servers, we assume that f_m servers are in Malicious Collusion (MC). Their aim is to create time-stamps with the same incorrect time. Each of the other $f_e = f - f_m$ servers independently delivers a fake time-stamp without colluding.
For a given document, active servers are randomly chosen in order to reduce the probability of DDoS: the attacker must attack the whole n servers ($n >> k$) to be sure to succeed. Following Requirement 4 , we assume that the number f of failed servers is bounded by $n/3$.
Let us now focus on a configuration where k servers time-stamp a given document. Each of these servers issues a time-stamping fragment and a vote allows them to obtain a certified complete time-stamp: a complete time-stamp is certified when more than $k/2$ servers propose the same date t. Of course, two servers may correctly time-stamp the document with two different but very close values. There exist many solutions to solve the problem of determining

t from a cloud of values. We may assume that two values represent the same date if they belong to a fixed range Δt. Another solution is to ask the client to time-stamp its document locally and submit this time-stamp for acceptance to the distributed system. These methods do not affect the security analysis of the scheme.
Let us now analyse the probability that an attack succeed, assuming that the servers are chosen in a uniform way. The various parameters of the general scheme are abreviated and given Table 1.

FS	failed servers
MC	failed servers which are in malicious collusion
n	number of available servers
k	number of active servers
f	total number of failed servers
f_m	number of MC servers
f_e	number of failed non MC servers ($f - f_m$)
d	number of active failed non MC servers

Table 1. Parameters of the configuration

A denial of service may occur as soon as more than $k/2$ active servers are FS.

The probality that at least $k/2$ active serveurs are FS is

$$P(FS \geq k/2) = \frac{1}{\binom{n}{k}} \sum_{k/2 \leq u \leq f} \binom{f}{u}\binom{n-f}{k-u}.$$

The following table gives examples of probabilities depending on different numerical values:

n	k	f	$P(FS \geq k/2)$
36	18 $(k \geq n/2)$	12	$3.7\ 10^{-2}$
36	12 $(k < n/2)$	12	$1.26\ 10^{-1}$

However, it may also be possible that more than $k/2$ active servers are in malicious collusion. In this case, $f_m > k/2$ and these active servers may contribute to forge a certified time-stamp.

The probability that exactly u active servers are in malicious collusion while d servers are active FS without colluding is

$$P(u,d) := \frac{\binom{f_m}{u}\binom{f_e}{d}\binom{n-f}{k-u-d}}{\binom{n}{k}}.$$

Figure 1 represents the continuous variation (by using the Gamma function instead of the factorial) of $P(u,d)$, $(0 \leq u \leq f_m)$ for $n = 36$, $k = 12$ $f = f_m = 12$ $(d = 0)$. We notice that the probability is maximal, equals to the value 0.3, for $u = 4$. In fact, a collusion attact succeeds only for $u > k/2 = 6$, and in this case, we have $P(6,0) = 0.099$, the other probabilities $P(u,0)$ for $k/2 <$

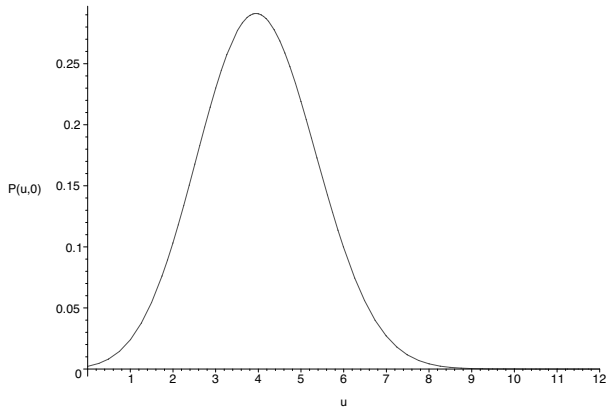

Figure 1. $P(u,d)$ **for** $n = 36$**,** $k = 12$ $f = f_m = 12$ **(**$d = 0$**)**

$u \leq f_m$ being decreasing and small $(P(7,0) = 0.026$, $P(8,0) = 0.004, \ldots)$.

The following table gives examples of probabilities $P(k/2,d)$ depending on different numerical values of parameters:

n	k	f	f_m	d	P
36	18 $(k \geq n/2)$	12	10	0	$1.4\ 10^{-3}$
36	18	12	10	2	$3.8\ 10^{-4}$
36	18	10	10	0	$3.4\ 10^{-3}$
36	12 $(k < n/2)$	12	7	0	$7.5\ 10^{-4}$
36	12	12	7	2	$5.9\ 10^{-4}$
36	12	12	12	0	$\mathbf{0.99}$
36	12	7	7	0	$2.6\ 10^{-3}$

In this table, the worst probability is close to $1/10$ and is reached when $k = f = f_m = 12$.

This scheme does not provide a satisfactory security in a context of denial of service attack. Moreover, considering the worst configuration, an attacker colluding with the MC servers could successfully backdate a document after a relatively small number of requests (about 10), without being discovered. This is due to the diffusion property of hash functions: a modification of one bit in the document leads to very different hashes. Therefore, the attacker may submit several times (almost) the same document until his request is processed by the MC servers.

5. A time-stamping scheme

In this section, we propose a time-stamping scheme which is not vulnerable to the attacks presented previously. It is composed of internal time-stamping boxes and

n servers managed by many independent entities. The idea of using distributed system to archive and sign documents has already been studied in [10]. Distributed time-stamping systems have also been studied, for example in [3] and [7]. However, none of these (rare) studies were able to design a secure and efficient system. In [Tak99], a scheme using distributed signing servers is proposed. However, this solution does not take into account the problem of denial of service. Here, we aim to present a secure and reliable distributed scheme to time-stamp documents. The main parties involved in the scheme are the following:

- **The client** needs to time-stamp some documents.
- **The box** is given to each client. Depending on the required security level, it can either locally time-stamp a document or randomly determine k active servers for a distributed time-stamping.
- **The network of servers** builds a time-stamp for each document, using time-stamping protocol $\mathcal{S}$.
- **The replicated database** serves as a publication media. Each server of the network records the time-stamps of all the clients. Therefore each server holds an entire copy of the time-stamp database.
- **The verifier** runs verification protocol $\mathcal{V}$ in order to check the correctness of the time-stamp of a given document.

Next paragraph describes in details this scheme.

5.1. The time-stamping scheme

Each client c has a calculation box B to which she submits the document D to time-stamp. Two levels of security are provided. The time-stamping can either be performed locally by the box thanks to classical protocol (level 0), or by the distributed system of n servers (level 1). The box

1. calculates the hashed value of D, denoted $h(D)$;
2. level 0 : locally time-stamps the document; end of the protocol.
   level 1 : determines randomly the k active servers;
3. signs the hash on behalf of the client c to form the request $r := (h(D))_c$;
4. sends the request r to the k servers and waits for an acknowledgment from each of them.

When level 1 is chosen, the document D is time-stamped in accordance with a scheme which can be described in the following way. Time is discretized in rounds of length Δt. Servers and clients are synchronized regularly and we do not take into account possible network malfunctions. Each round is identified by an absolute date. For example, a round can be identified by: January 1^{st} 2005 at 9.05am. During a round t, each server receives a number of requests which is approximately the same if this number is large enough, since active servers are chosen randomly.
Suppose that the server S_i receives p_i requests. Let T_{S_i} denote the array formed by the p_i requests during the round identified by t. At the beginning of the next round (identified by $t+1$), the server S_i signs the concatenation of t with T_{S_i} to obtain $CS_{i,t}$. This is the stamp of the server S_i for the round t. Define $CS_{i,t} := (T_{S_i}||t)_{S_i}$. This stamp is then broadcasted to every node in the network. Hence, each server knows the list of all the distinct x requests $r_1, \ldots, r_x$, which have been submited during round t.

At the beginning of the next round (identified by $t+2$), the server S_i calculates the **global time-stamp** of the round t, CG_t and records it in its database. CG_t serves as a published value and is defined using one way accumulator functions that we have to define first.
Let Λ and E be two sets and define a family of maps

$$\begin{aligned} T_y : \quad & E \to E \\ & x \mapsto \ T_y(x) \end{aligned}$$

with $y \in \Lambda$, and so called dual maps

$$\begin{aligned} {}_xT : \quad & \Lambda \to E \\ & y \mapsto \ {}_xT(y). \end{aligned}$$

We assume that

i) ${}_xT$ is a one way function;

ii) for all a and b in Λ, $T_a \circ T_b = T_b \circ T_a$.

The map $F : E \times \Lambda \to E$ defined as $F(x,y) := T_y(x)$ is an accumulator function in the sense of [4]. We denote by T_{ab} the composition $T_a \circ T_b$.
The global time-stamp CG_t of the round t is defined by

$$CG_t := h(T_\rho(t)),$$

where $\rho := r_1 \ldots r_\nu$. For each of its clients in the round t, the server calculates and signs the **client time-stamp**, which is composed by the following values:

$$r, \ t, \ CG_t, \ CG_{t,r} := T_{\rho_r}(t)$$

where r is the request of the client to whom the stamp is to be sent, t is the round identifier, CG_t is the global stamp and $\rho_r := r_1 \ldots r_{r-1} r_{r+1} \ldots r_\nu$. Hence, $CG_{t,r}$ is the accumulation of all the requests but r.

For a given document, each box receives k stamps. If all the requests are received by the server during round t,

then all the k stamps are the same. However, it may be possible that some servers receive the request during round t while some others receive it during the round $t+1$. This situation does not constitute a problem. Indeed, the proof that the document has been time-stamped during round t can be done as long as at least one client time-stamp has been created.

Note that the value Δt has to be carefully chosen, taking into account different parameters like the properties of the network or the method of synchronization which may be time consuming.

5.2. Verification scheme

Server databases are to be consulted by verifiers. Each database record consists of the values CG_t, r, and t.
The verification of the time-stamp is as follows:

1. The request r has been involved in the construction of the global time-stamp if
$$CG_t = h(T_{\rho_r} \circ T_r(t)).$$
2. If required, the verifier can check that CG_t is the published value corresponding to the round t.

5.3. Robustness of the scheme

Attacks can be arranged into two categories:

- Attacks performed on existing time-stamps.
- Attacks performed during the construction of time-stamps.

The first type of attack consists in modifying the stamp while keeping its provability property. The success of such attacks depends on the robustness of the cryptographic functions that the system uses. The choice of the cryptographic functions is essential since time-stamps are to be valid for a long time (a few years).
Let us now study the robustness of our scheme against attacks of the second category. The number of failed servers being less than $n/3$, an audit is always able to detect either an error or an attack.
Backdating is impossible since the time-stamp would not be provable.
Postdating (which can be seen as a form of denial of service) is possible. It requires that the k active servers, in malicious collusion, wait during λ rounds before handling the request. The probability of such an attack is not negligible. It is equal to $P_{n,k} := \binom{n/3}{k}/\binom{n}{k}$. With the following parameters: $n = 24$, $k = 5$, we obtain a probability of $P_{24,5} = 1.3\ 10^{-3}$. In order to reduce the consequence of this attack, the client is requested to send again the time-stamping request to k new active servers when no acknowledgement has been received after a time of Δt. In this case, time-stamping is performed after a delay of Δt and the precision of the time-stamp is reduced. This solution also holds for a pure DoS attack. This study is resumed in Table 1, where the last column represents, in case of successful attack, the difference between the correct date and the date of the time-stamp.

Attack	Possibility	Detection	Probability	Error
Antidating	NO	–	–	–
Postdating	YES	YES	$< \binom{n/3}{k}/\binom{n}{k}$	$+\Delta t$
DDoS	YES	YES	$< \binom{n/3}{k}/\binom{n}{k}$	$+\Delta t$

Table 2. Robustness of the scheme

Our protocol makes use of three types of cryptographic functions. Hash functions, signatures, and accumulators. Accumulator functions may be a simple modular exponentiation. In this case, we have $T_r(x) := x^r \mod s$, where the parameter s is an RSA modulus, and can be defined according to the recommandations of [4]. In particular, we must have $(r, \phi(s)) = 1$. Notice that ${}_xT : r \mapsto x^r \mod s$ plays the role of a one way function since finding r is finding the discrete logarithm of ${}_xT(r) = x^r \mod s$ which is known as a hard problem. Moreover, the property $ii)$ also holds since $x^{ab} = x^{ba} \mod s$. With the previous notation, $CG_t := h(t^{\prod_{j=1}^{\nu} r_j} \mod s)$ and $CG_{t,r} := t^{\prod_{j=1}^{\nu} r_j / r} \mod s$. Hence the equation used for verification is now $CG_t = h((CG_{t,r})^r \mod s)$.
However, public key algorithms of RSA type being not efficient, accumulators based on this technique may not represent a viable solution. We recommand to use new algorithms like XTR (Efficient Compact Subgroup Trace Representation) or algorithms based on elliptic curves. In the case of elliptic curves, we use an additive group instead of a multiplicative group. We have $T_r(t) := r.t$, where t is a point of the curve and r an integer. In this case, accumulators and signatures may use common parameters (same curve, same field, ...) in order to simplify the scheme. This study, furthermore very interesting, is outside the scope of this paper.

6. Random generators

Our scheme requests a generator to randomly select k distinct elements from a set E of n elements. Hence, we seek for a uniform generator, cryptographically secure and which satisfies some requirements particularly on the time of execution but also on the memory space.
There exist severals generating algorithms. The simplest way to build such a uniform random generator is to select an element $a_1 \in E$ and then select an new element, distinct

from the preceding one, and repeat the process until we get k elements. We consider two classical cases:

- Each selected element is dropped from the set E. Consequently, we have to build k random generators of q symbols, with $n - k + 1 \leq q \leq n$. This leads to a possible bias if we use a binary source generator.
- Selected elements are kept in the set. In this case, only one generator is used and the problem of possible bias concerns just this generator. However, it may take a lot of draws before we obtain k elements since in our model, k is not very small compare to n. Therefore, the probability to draw an already selected element is not negligible. Recall that the average number of draws is approximately $n \log n$ (see for example [9]).

One of the most famous generating algorithm is probably RANKSB of H. Wilf (see [14] for details). Its execution time, in average, is $\mathcal{O}(k)$, when $k << n$. But in our case, it will be around $\mathcal{O}(n)$ and execution time is not constant.

Now, we propose an algorithm based on the notion of random walks on a finite group G. We refer the interested reader to the basic surveys of N. Sloane [15] and D. Aldous [1]. Let $Q^{(m)}$ be the distribution on G determined by the issue of the walk after m steps starting from the identity element. The initial distribution is Q (that is, the probability to go from g to h in the group is $Q(g^{-1}h)$). Let U be the uniform distribution on G. When $Q^{(m_0)} > cU$ for an integer m_0 and a constant $c > 0$ (mixing case), the total variation distance $d(m) := \max_{A \subset G} |Q^{(m)}(A) - U(A)|$ between $Q^{(m)}$ and U tends to 0 at an exponential rate. But this majoration is generally not numerically useful. Consider the case where G is the permutation group of E. We choose the walk represented by the mixing of a deck of n cards by the following method: insert equally likely the topmost card within the deck. From [2] (Theorem 1), $d(n \log n + \gamma n) \leq e^{-\gamma}$ for all $\gamma > 0$ and $n \geq 2$. Let us apply this result for $n = 24$. An explicit version of Stirling formula leads to the concrete bound

$$d(1393 + 17s) \leq \frac{1}{2^s 24!}$$

for any integer $S \geq 0$. This method does not provide us with a uniform generator, but the bias, represented here by the quantity $d(m)$, can be made negligible by using enough computing power. Moreover, the fluctuation of performances of the binary source generator can be corrected by increasing the number of iterations.

7. Conclusion

Our scheme represents an alternative solution to classical monoserver schemes. The level of security and reliability can be achieved by carefully adjusting the k and n parameters. The distributed publication allows us to avoid a costly publication in a (not electronic) newspaper.

This scheme also offers the possibility to time-stamp documents in an off-line mode. In that case, the system may locally adopt a classical linking scheme and the publication would be done by the n servers when on-line. It may find applications where clients are not to be continuously connected to the system like, for example, in spontaneous networks.

We think that multiserver schemes represent the right way to obtain efficient solutions in the domain of time stamping. Moreover, we think for further work that the use of modern cryptographic tools, like for example threshold cryptography, may help to simplify protocols and avoid interactions between servers.

References

[1] D. Aldous. Random walks on finite groups and rapidly mixing markov chains. *Séminaire de Probabilités XVII*, 1981.

[2] D. Aldous and P. Diaconis. Shuffling cards and stopping times. *Am. Math. Monthly*, 1986.

[3] J. Benaloh and M. de Mare. Efficient broadcast time-stamping. *Technical Report 1, Clarkson University Department of Mathematics and Computer Science*, 1991.

[4] J. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures. *Advances in Cryptology–EUROCRYPT'93*, 1993.

[5] A. Buldas, P. Laud, H. Lipmaa, and J. Villemson. Time-stamping with binary linking schemes. *Advances in Cryptology–CRYPTO '98*, 1998.

[6] A. Buldas and H. Lipmaa. Digital signatures, timestamping and the corresponding infrastructure.

[7] S. Haber and S. W.S. How to time-stamp a digital document. *Journal of Cryptology*, 1991.

[8] M. Just. Some timestamping protocol failures.

[9] D. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, 1981.

[10] P. Maniatis, T. Giuli, and M. Baker. Enabling the long-term archival of signed documents through time stamping. Technical report, Computer Science Department, Stanford University, 2001.

[11] H. Massias and J. Quisquater. Time and cryptography. Technical report, Universit'e catholique de Louvain, 1997.

[12] H. Massias, X. Serret, and J. Quisquater. Timestamps: Main issues on their use and implementation. *In Proceedings of IEEE 8th International Workshops on enabling Technologies*, 1999.

[13] R. Merkle. *Secrecy, authentication, and public key systems*. PhD thesis, Electrical Engineering Department, Stanford University, 1979.

[14] A. Nijenhuis and H. Wilf. Combinatorial algorithms for computers and calculators, 1978.

[15] N. Sloane. Encrypting by random rotations cryptography. *Proceedings of the Workshop on Cryptography*, 1983.