Timestamps: Main issues on their use and implementation

H. Massias, X. Serret Avila, J.-J. Quisquater
Université Catholique de Louvain
Crypto group
Place du Levant, 3
B-1348 Louvain-la-Neuve, Belgium
massias, serret@dice.ucl.ac.be
jjq@timestamping.com

Abstract

This paper discusses some of the issues that appear on the creation and use of secure digital timestamps. From the timestamp creation side, it first introduces a review on the existing systems, giving a recommendation for the one we believe is more suitable for the general use. Afterwards, it deals with an extension of the basic scheme in order to be used in the enterprise environments. Finally, it discusses the general scalability problem, analyzing an existing system and proposing a more adequate solution. From the timestamp use side, it discusses one of the possible misuses and it proposes a solution to secure it.

Key-words: secure digital timestamping, trust, scalability, trusted third party.

1 Introduction

The date of digital documents creation and the times expressed in them are becoming increasingly important as digital documents are being introduced into the legal domain.

We define digital timestamp as a digital certificate intended to assure the existence of a generic digital document at a certain time.

In order to produce fully trusted timestamps, very specific designs have been introduced. We give an overview of the most relevant methods and we introduce the one we used for the implementation of the Belgian project TIMESEC (`http://www.dice.ucl.ac.be/crypto/TIMESEC.html`), justifying our choice for it.

From that starting point, we discuss some of the issues we detected on using timestamps. Afterwards, we analyze their use in an enterprise environment and we give some hints on how to further scale the basic system design. Finally, we discuss the implications of the system design and implementation on the trustworthiness and accuracy of the timestamps obtained.

2 The timestamping methods

There are two families of timestamping techniques, those that work with a trusted third party and those that are based on the concept of distributed trust. Techniques based on a trusted third party rely on the unpartiality of the entity that is in charge of issuing the timestamps. We can also classify those techniques into two different types: those where the third party is completely trusted and those where it is partially trusted. The technique based on the distributed trust consists of making documents dated and signed by a large set of people in order to convince the verifiers that we could not have corrupted all of them. A detailed study of timestamping techniques can be found in [5]. Nowadays, all the commercial implementations of timestamping services are based on the first approach. It is for this reason that we will concentrate on the trusted third party approach.

The "easy" solution, which consists of concatenating the document with the current time and sign the result has been discarded because it has two main drawbacks:

1. The first one is that we must completely trust the Secure Timestamp Authority (STA), which can issue undetectable back-dated timestamps.
2. The second one is related to the limited lifetime of cryptographic signatures, which can be shorter than the document time-to-life. Below there is a detailed discussion about that fact (see section (3.3)).

The timestamping method that we have chosen uses a binary tree structure and has been described in [3]. This

method works by rounds. For each round a binary tree is constructed with the requests filled during the round. In Figure 1 we can see a graphical representation of a round constructed using this method.

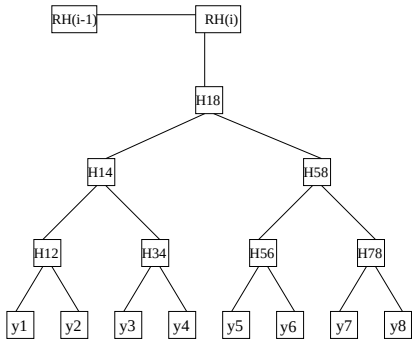

Figure 1. The binary tree structure

Each of the requests consists of a hash value of a given document. The leafs of the tree are each of those hash values. The leaf values are then concatenated by two and hashed again to obtain the parent value ($H_{34} = H(y_3 \mid y_4)$). This process is repeated for each level until a single value is obtained.
The top value of the round tree (H_{18}) is then concatenated with the value obtained for the preceding round (RH_{i-1}) and finally hashed again to obtain the actual round value (RH_i).

The timestamp of the document contains all the values necessary to rebuilt the corresponding branch of the tree. For example, the timestamp for y_4 contains $\{(y_3, L), (H_{12}, L), (H_{58}, R), (RH_{i-1}, L)\}$. The verification process consists of rebuiding the tree's branch and the linking chain of round values until a trusted (for the verifier) round value is recomputed. This verification method is explain in detail in [3], [5] and [4].

Periodically, one of the round values is published on an unalterable and widely witnessed media (Ex: newspaper...). These special round values, which we call "big round values" are the base of the trust for all the timestamps issued. All verifiers must trust these big round values as well as the time associated with them. This is a reasonable requirement because those values are widely witnessed. The absolute time trusted by all the potential verifiers is the time indicated by the unmodifiable media. For the following discussions we suppose that this time is the same than the time indicated by the STA for the big round. Another requirement is to force the clients to check the timestamps as soon as they get them. In that way the process is continuously audited and the STA will not have any margin to manoeuvre in an untrusted way.

A very useful method for extending the lifetime of timestamps is described in [1]. It basically consists of re-timestamping the hash of the document as well as the original timestamp before the hash function is broken.

3 Scope of a timestamp

3.1 What trusted information does a timestamp provide?

As we explained above, the timestamp of a document is in fact the timestamp of the hash of the document. It is secure because there is a strong one to one relationship between a document and its hash, and in that way the timestamping authority will not be able to gain any knowledge of the data it timestamps. But this advantage has also a drawback, i.e. the STA will guarantee nothing else but the time of the reception of the hash. The interpretation and exploitation of the timestamp is delegated to the verifier, who must be aware of what the timestamp on a hash value really means. If the timestamped document is the result of a computation, then the timestamp does not guarantee anything about the original data used for this computation, as we will show for signature in section (3.2). Before giving the timestamp a significance other than the time of submission of a document hash, the verifier must study its "context environment" carefully.

If we want to guarantee the identity of the submitter, the STA must first authenticate it, otherwise it has no significance to include it in the timestamp. Checking the identity is not a normal feature of a STA, but could be done by a notarisation authority.

With a timestamp we can prove that a document existed before a certain time, but if someone steals this document and then timestamps it, he will also be able to prove the same thing. In some cases (patents ...) it is necessary to prove the ownership as well. One way to resolve this problem could be to timestamp a signature of a document. This will prove that the signing identity was in possession of the considered document at the indicated time. However this solution must be used with extreme care as shown in section (3.2).

As timestamps only contain hashes, in isolation they cannot prove anything about the original document (like format, contents, references to other documents ...).

3.2 Timestamp of a signature

We can believe that the timestamp of a document signature can also be considered as a timestamp of the document. We will show that in some cases, it is not.

What follows is illustrated in Figure 2. If we have the timestamp of the hash of a signature: $Timestamp(H(S))$, then the value S is fixed in the time. The question now is the following: Is $Timestamp(H(S))$ also a valid timestamp for the document Doc?

As we said, S is fixed, but in fact S alone is just a data stream that has no intrinsic significance. In other terms,

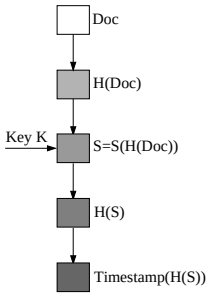

Figure 2. Timestamp of a signature

we can search to bind a significance to it other than the original one. The trick here is to say that S is in fact the signature of another document Doc'. If we can prove that, we will have then a valid timestamp $Timestamp(H(S))$ for another document.

To resume the problem: if we consider the timestamp of a signature as a valid timestamp for the signed document then chances are high that we will be able to backdate documents.

3.2.1 A scenario of the attack

We make the following hypothesis:

- the timestamped signature does not contain the public key necessary for the verification
- the certificate on the forged signing public key does not contain a "not valid before" field
- we accept the timestamp of a signature as a valid timestamp for the signed document.

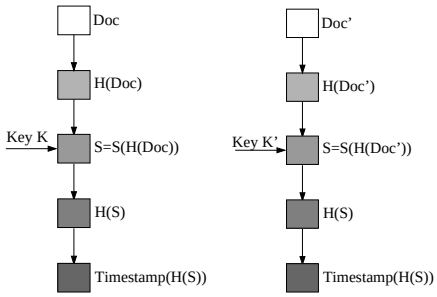

Figure 3. The attack

Above is the skeleton of the attack, which is illustrated in Figure 3. We dispose of a stream S that has been timestamped at time t, and we want to prove that a document Doc' that we have created at time $t+\delta$ was made at time t. For this purpose, we hash Doc' and then we search for a key K' such that the signature made with this key on Doc' will be equal to S. We will then have obtained a valid timestamp $Timestamp(H(S))$ for Doc'.

Now the problem is reduced to the problem of finding a key such that the signature of a document that we choose is equal to a fixed data stream.

3.2.2 Attack against RSA signature

Let d be the private key and (e, n) be the public exponent and public modulus. m is the message that we want to backdate. m could eventually be the message already padded and eventually with redundancy (ISO9796, PKCS) . Now the problem is to find d and n so that

$$m^d \text{mod } n = S.$$

If n is fixed, this is the discrete log problem, which is hard. But, as we can choose n, we could take $n = pqrs...$ so that the discrete log problem will then be easy. Nobody will be able to check that n is a well constructed modulus. Once we have obtained d, it is easy to calculate e because we know the factorization of n. The final step is to register the key (e, n) in the usual way.

3.3 PKIX, CRL, Certificate

IETF's PKIX working group uses signature timestamping in order to provide "non-repudiation" of digital signatures. The solution they propose for the timestamping problem can be found in the working document entitled "Time Stamp protocols" (September 23, 1998). It consists of having a fully trusted STA, which concatenates the date to the hash (which is the hash of the signature in our case) submitted and then signs the result. All the possible verifiers must completely trust the STA because there is no way to check the integrity of the STA. Another drawback of this system concerns the limited lifetime of electronic signatures. Signature algorithms rely on keys, which have a lifetime a lot smaller than crypto-algorithms. One key can be compromised without compromising other keys or the signature algorithm by itself, this is one of the reasons to use certificate revocation lists. As the signature algorithms without recovery use hash functions, we can assert that the time-to-life of a hash is, at least, as long as the time-to-life of a signature (actually, is much longer). This illustrates the fact that using a timestamping protocol that does not rely on any secret information, like the one described in section (3.4), is stronger than the trivial "add date and sign" protocol.

3.4 Solutions for timestamping signatures

As explained in [2], a good solution is to timestamp the combination of the document and its signature. Another

solution could be to timestamp the signature together with the signing key. These are solutions for counterfeiting the attack described in section (3.2.2), but they do not solve the problem of certifying the time and ownership together. For this last purpose, the best solution is to have a notarisation entity that will certify (and first verify) that Mr. X or Company Y did own the document corresponding to the hash H at time t. In a PKIX framework, the certificate of the signing key must be timestamped as soon as possible after the signature on a document has been issued. As mentioned in [6], for long term valid signatures "all certificates, cross-certificates, CRLs or OCSP responses must be timestamped in order to protect against the later compromising of a certification key". This implies a considerable overhead as this process must be repeated for each signature.

4 Possible use of timestamping in a company

The utility of digital timestamping in a company can be illustrated by multiple examples:

- Imagine that someone is able to change the date of a company backup without leaving any trace of this modification. In that case the backup will be completely useless and also dangerous. Unfortunately, changing the date put by the operating system or by a software is just a matter of gaining access to the document. If we securely timestamp this backup, we will at least be able to detect the modification of the backup date, minimizing possible harms.

- Time forgery in database entries can also be used as an attack. This is specially true when distributed databases are used and not all of the database engines can be fully trusted; because, as an example, their physical environments cannot be secured up to the same level.

We propose now an example of a timestamping infrastructure for a company.

For the following discussions we use "external" and "internal" words in respect to the company.

4.1 Example of a proposal

We separate the company documents into two groups:

- the documents that must be communicated to the outside of the company (internal and external signification),

- and the documents that will only be used for inside purposes (internal signification).

These two types of documents do not require the same level of trust for their timestamps. We propose a scalable solution that treats these two kinds slightly differently . The problem is that the employees are not always able to determine the kind of the document they wish to timestamp at the submission time. So our proposal also permits to use a timestamp issued for the second purpose in the first purpose scope, with the drawback of loosing some precision.

We propose to have a local STA using the timestamping technique described in section (2). The big round publishing consists of making the big round value to be timestamped by an external STA. This external STA issues timestamps trusted by the possible external verifiers. When an employee wants to timestamp a document for an external use, he will notify the internal STA, which will make the corresponding round value timestamped by the external STA. To summarize, two kinds of round values will be made externally timestamped: the big round values and the round values containing a document for which the submitter has asked for external significance.

We indicate now what a verifier can trust in such timestamps by separating the different kinds of verifier (see Table 1). This allows us to justify our proposed design. "Absolute time" means a date like March 4,1999, 16:20' GMT. "Relative time" means that we determine that a document has been timestamped after or before another one. As indicated in section (2), a working timestamping system requires that all verifiers trust the publishing entity. In our case, every actor must trust the values published on an unmodifiable media by the external STA or we are not able to prove anything.

Level	Trust-able precision
absolute time vs. exterior	external STA big round
relative time vs. exterior	external STA round
rel. and abs. time vs. exterior (STA+)	external STA timestamp
absolute time vs. interior	company STA big round
relative time vs. interior	company STA round
rel. and abs. time vs. interior (c.STA+)	company STA timestamp

STA+ means that the external STA is trusted
c.STA+ means that the company STA is trusted

Table 1. Trust level

A justification for Table 1 is the following: an external verifier does not trust the local STA. If he does not trust the external STA, but just the published round values (which is the normal case), he will only trust the absolute time put by the external STA for big rounds because the time for the intermediate rounds is put by the external STA itself. For relative comparison of two timestamps, the precision is the round, because he is able to check the unmodifiable chain between two round values. However he is not able to

compare two timestamps issued during the same round if he does not trust the STA for not re-ordering the requests inside a given round. The same explanation fits for a local verifier. The sole difference is that the local verifier trusts the time put by the local STA for the published round values. In other words, he completely trusts the external STA for timestamping correctly the round values given by the local STA. This requirement is somewhat smaller than trusting the external STA for all the timestamps that it issues.

We explain now how to externally use a timestamp that was originally issued for internal use . If we provide the external verifier with the elements of the chain that links an internal request with the next published round value (speaking in terms of the internal STA), then the verifier will have the same trust in the internal timestamp than in a timestamp originally issued during the published round. Unfortunately, as we can see, the precision is lowered. This illustrates our choice of making a round value externally timestamped when the submitter knows in advance that it will be used for external verification.

5 Using multiple STA's (scalability)

The problem of scalability is a major issue for timestamping techniques. The solution of using multiple STA's does not provide the accuracy and trustworthiness of just one STA, but it is still feasible without being forced to fully trust all the STA's.

One way to be able to scale a timestamping system is to have different levels of STA. STA's on higher levels concentrate the round values of the STA's on lower levels. Unfortunately, we will see that this cannot be scaled to the infinity and that trade-off must be found. Another complementary way is to define comparison rules for the values published on different unmodifiable media. Everybody must also trust those unmodifiable media. The more STA levels, the less unmodifiable medias we need. However the more STA levels we have, the less precise the timestamps are, as we have illustrated in an example in section (4). On the other hand, if there are a lot of unalterable medias, the trust needed in the system becomes enormous because all the possible verifiers must trust all the unmodifiable medias. The comparison rules are also more difficult to state. A trade-off must be found between the number of STA levels and the number of unmodifiable medias.

We could also define several types of timestamping infrastructures that use several levels of STA's.

1. The first type uses the lower levels of STA's just as "concentrators". Only the top level STA will issue the timestamp. This is the method use by the Surety software (`http://www.surety.com`). We can also imagine two variants of this system.

   (a) The STA at level $i+1$ trusts the STA at level i for putting the right time on its round subtree. A STA collects all the round subtrees corresponding to the time t and computes a new round subtree that it dates again with t. Then it submits it to the STA just one level higher than it is. If it is in the highest possible level, the STA will publish the round value that it obtains. Finally, it will issue all the timestamps for all the documents submitted to all the STA's. To achieve that, the lower STA's do not only submit the round subtree value to the higher STA, but they give the entire round subtree.

   (b) A variant of this method is that the higher STA's does not wait for the values of the corresponding round subtrees from the lower STA's as above. They consider their own round, which they date with the actual time. As we can see, this implies a delay proportional to the number of levels. The major drawback of this solution is that the precision of the absolute time given is low. The advantage is that fewer trust between the STA's is needed. Still, the lower STA's must trust the higher STA for correctly issuing the timestamps.

   The difference between these two variants and the technique of using only one STA is very small (it lowered only lightly the work of the top level STA) and the scalability is then reduced. The two main drawbacks of this technique (unverifiability and delay) makes us think that it is not a good solution.

2. The other possible method is the one that we described above (see section (4)). This method can also be scaled to more than two levels. The trust in the different STA's is lowered and, as we explained, different level of trustable precision can be achieved. The sensible point is to find a trade-off between the number of unmodifiable media publishers and the number of levels of STA's.

6 Issues to think about when designing a concrete system

The remarks that we make in this section come from our experience on designing and implementing a complete timestamping system (see [4]).Our principal design requirement was to minimize the trust required on the third party issuing the timestamps. We use the technique presented in (see section (2)) with the small difference that we build two trees in parallel for each round using two different hash functions. This allows to remain secure in

case of an unexpected break of one of the hash functions used.

The client submits two hashes of the document he wants to timestamp using the same hash functions as the one used by the STA. The STA has no knowledge of the kind of document that is being timestamped. It also can not check that the two hashes received have been computed from the same document or with the supposed hash functions. If this is not the case, the timestamped document will not be validated by the validation process, but this can not be detected in advance.

As we explained in section (3.3), signatures should not be use for the timestamping process. In our design we only use it for STA authentication purposes when sending back the timestamp to the client.

A good timestamping technique should be auditable. In our design, all the computations of the STA can be checked at any time. All the values timestamped, as well as the round values, are logged. With these informations any outside entity can recompute and check all the round values issued by the STA.

As you may have seen in section (2), the binary tree is defined for a number of leafs (request) that is a power of 2. In general,this will not be the case . We could create fake requests to finish the tree, but it will add a lot of requests. Imagine that we have $2^n + 1$ requests, we need then to add $2^n - 1$ fake requests. A smarter solution is to add a random value only when needed. We add at most n values (one for each level of the tree). We call this node "special node". Another solution could be to use the 0 value or a fixed value instead. It is as secure as the solution we use if the hash functions are "perfect". As hash functions are only "presumably perfect", we though that we could make our design more secure with really few additional computations.

In our implementation the STA queues the requests, and computes the tree at the end of the round. At first sight, it could seem a better and more secure solution to build the tree as soon as the requests arrive and at the end of the round finish the computation of the tree by getting the last round value. In fact, this solution is harder to implement, and has no effect on the security because no one can check that the STA does not perform any reordering of the requests before it publishes the round value.

Our design uses a round queue for each round. There is a different thread assigned to each round queue, which is mainly waiting for the end of the round. Once the round is closed, the thread wakes up and constructs the round tree finally obtaining the round value. The time indicated in the timestamp is only determinated when the request is put in the round queue. The round queue does not accept any request after the end of the round, in that way, the computation of the tree can begin immediately. Another solution could be to determine the time as soon as the request is received. However we detected during the implementation design that it will then be difficult to know which are the requests belonging to a round waiting to be processed once the time for the round has expired. On the other hand, if there is an abnormal delay between the reception of a request and the queuing, then the computation of the tree and then the issuing of the timestamps will be delayed.

The values timestamped as well as the round values are logged on a file. This feature permits to define an audit process.

When checking the validity of a timestamp, the verifier asks the STA for the necessary values. These values are all the round values between the last and next big rounds. With this material, the verifier will be able to check the validity of the timestamp without having to trust the STA at all.

7 Conclusion

As we explained, the problem of securely timestamping documents is not as easy as it seems. The lifetime of the crypto-algorithms used to define a timestamping system must be carefully studied. A value associated with a time must be published somewhere in an unmodifiable media in order to be trusted by all the possible verifiers. The place and frequency of this published round value influences directly the trust and granularity provided by the timestamps. The dependency on this published value is the major brake for the scalability of the timestamping system, which must be further investigated.

References

[1] D. Bayer, S. Haber, and W.-S. Stornetta. Improving the efficiency and reliability of digital timestamping. In S. Verlag, editor, *Sequences'91: Methods in Communication, Security, and Computer Science*, pages 329–334, 1992.

[2] S. Haber, B. Kalisky, and S. Stornetta. How do digital timestamps support digital signatures? *Cryptobytes*, (3):14–15, autumn 1995 1995.

[3] S. Haber and W.-S. Stornetta. How to timestamp a digital document. *Journal of Cryptology*, 3(2):99–112, 1991.

[4] H. Massias, X. S. Avila, and J.-J. Quisquater. Design of a secure timestamping service with minimal trust requirements. accepted at the 20th Symposium on Information Theory in the Benelux, May 1999.

[5] H. Massias and J.-J. Quisquater. Time and cryptography. Technical report, TIMESEC Project (Federal Governement Project, Belgium), 1997. available at `http://www.dice.ucl.ac.be/crypto/TIMESEC.html`.

[6] H. Nilson and D. Pinkas. Validation of electronic signatures, white paper. available at `www.openmaster.com/whitepapers/es_validation.pdf`, January 1999.