THE DIGITAL TIMESTAMPING PROBLEM

Bart Van Rompay*, Bart Preneel†, Joos Vandewalle

Katholieke Universiteit Leuven, ESAT–SISTA/COSIC
K. Mercierlaan 94, B-3001 Heverlee, Belgium
bart.vanrompay@esat.kuleuven.ac.be

In this paper we examine the relation between time and cryptography. Timestamping systems provide temporal authentication of electronic documents, they certify the date a document was created or last modified. Important applications are found in legal situations, electronic contracts, the protection of Intellectual Property Rights, and Internet security in general. We give an overview of the problem and discuss several state-of-the-art techniques.

INTRODUCTION

The interaction between time and cryptography is a relatively new subject. [HS91] is the reference work in this area. A study was made in the framework of the Belgian Timesec project (see [MQ97] and [PRQ+98]). Most cryptographic research has been focused on secure messaging systems. Symmetric and public key encryption are used for protecting confidentiality. Authenticity (data integrity and data origin authentication) can be obtained by applying keyed hash functions (for the symmetric key case) or digital signatures (in the public key scenario).

However, the secure maintenance of documents with a long lifetime is more complicated. Key pairs used in public key systems have a limited lifetime and may be revoked if the private key has been compromised. In a non-repudiation service we must be able to determine, at a later time, if a document was signed (using the owner’s private signature key) within the validity period of the certificate. In other applications the time itself can be important, for example the date of an inventor’s patent claim to establish precedence over competing claims.

Digital timestamping provides the solution to this problem, by certifying the date and time on which a particular document is submitted to the timestamping

* Sponsored by the Timesec project of the Federal Office for Scientific, Technical and Cultural Affairs (OSTC), Belgium.
† F.W.O. postdoctoral researcher, sponsored by the Fund for Scientific Research – Flanders (Belgium).

service by the owner. When dealing with electronic data it will be necessary to
timestamp the data itself, without any regard to the physical medium on which it
resides. The security depends on the following general properties:

- It must be infeasible to timestamp a document with a date and time differ-
  ent from the present one.
- It must be infeasible to change even a single bit of a stamped document
  without the change being apparent.

A SIMPLE CENTRALIZED SERVICE

A basic solution for timestamping relies on the use of a trusted third party, the
Time Stamp Authority (TSA). The TSA appends each request with the current
date and time and digitally signs the result to produce a timestamp. The scheme
can be enhanced by using a hash function to compress the document sent to
the TSA. This allows to preserve the confidentiality of the original data, reduces
the bandwidth and storage requirements, and improves the efficiency (digitally
signing is a computationally expensive operation).

If a document is to be securely and uniquely represented by its hash value, the
hash function used must meet certain cryptographic requirements. This means
that the choice of function has to be imposed by the TSA. More specifically the
hash function needs to be one-way (impossible to invert the hash to find the
original data) and collision-free (impossible to find two documents with equal
hash values). These properties cannot be proven for practical functions and
many constructions have been broken (for example MD4, which was suggested as
a practical hash function in [HS91]). See [RPV98] for a discussion on practical
hash functions. RIPEMD-160 and SHA-1, recently standardized by ISO/IEC,
are good candidates.

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

1. Client A sends his identity and the hash value of the document he wants to
   have timestamped: he sends $ID_A, h(X)$.
2. The TSA returns the timestamp certificate $C = S_K(ID_A, h(X), n, t)$, where
   $n$ is the certificate serial number, $t$ the current date and time, and $S_K$ the
digital signature by the TSA.
3. The client receives the certificate and checks that it is signed by the TSA,
   that it contains the hash of the document he asked a timestamp for and the
correct time (within reasonable limits of precision).

Appeared in Proceedings of the 20th Symposium on Information Theory in the Benelux,
A. Barbé, E. C. van der Meulen, and P. Vanroose (eds.), Werkgemeenschap voor Informatie-
Another user who questions the validity of the timestamp $C$ for the document $X$ will use the following Verification Protocol:

1. Check that the digital signature on the certificate corresponds to the TSA.
2. Check that the hash value $h(X)$ corresponds to the document $X$.

The fundamental problem with this scheme is that we need absolute trust in the TSA since nothing prevents it from producing fake timestamps. One possible alternative is to distribute the trust over several randomly chosen signers, making forgery difficult since all parties involved would have to be corrupt ([HS91]). Such a scheme doesn’t even need the use of a centralized TSA. However the large degree of cooperation which is required from other parties seems to make it impractical in large environments. Hence we will further concentrate on schemes which enhance the centralized service by means of a mechanism that links together all the timestamps issued by the TSA, creating an unforgeable temporal chain.

A NOTE ON SECURE TIME

The timestamps issued by the TSA contain an absolute temporal measure, so there is a need for a common reference time, agreed upon by all users. Reliable time can be provided by atomic clocks and distributed by systems such as NTP, the Network Time Protocol, a time service offered on the Internet (see [NTP]), allowing the synchronization of internal clocks with an accuracy of about a few tens of milliseconds on a WAN.

THE BASIC LINKING PROTOCOL

In [HS91] a protocol is introduced that minimizes the trust requirement by making the TSA link all the issued timestamps together in a temporal chain. This is achieved by including in each timestamp information from the previous timestamp. To protect the temporal ordering we need another one-way and collision-resistant hash function $H$. The timestamps are constructed as follows:

1. The client, who makes the $n^{th}$ request to the TSA, sends $ID_n, h(X_n)$.
2. The TSA returns the certificate $C = S_K(ID_n, h(X_n), n, t_n, L_n)$, which contains linking information $L_n = (ID_{n-1}, h(X_{n-1}), t_{n-1}, H(L_{n-1}))$.
3. When the next request is received the TSA also sends $ID_{n+1}$ to client $ID_n$ to produce a link in the future direction of time. The complete timestamp consists of $(C, ID_{n+1})$.

4. Client $ID_n$ verifies the signature, the hash of his document and the time.

Someone who wishes to verify the timestamp will:

1. Check the signature of the TSA and the hash of the document.
2. To make sure there was no collusion between client $ID_n$ and the TSA he can contact clients $ID_{n+1}$ and $ID_{n-1}$, ask for their certificates and check the linking between them. Using their timestamps he can go as far forwards or backwards along the chain as he wishes.

The use of the hash function $H$ in linking timestamps together gives us reliable relative temporal information. One can check that a particular timestamp was created before or after another one. Users of the system could generate and timestamp random numbers at regular time intervals to obtain trusted linking items so they can locate other timestamps between them. Another possibility is periodic publication of timestamps in authentic media such as newspapers.

If the TSA cannot find collisions for the hash function $H$ it cannot forward-date or back-date timestamps along the chain. A possible attack would be by producing a fake chain of timestamps which is longer than a suspicious challenger will verify. In [Jus97] an attack is suggested that generates a fake sub-chain. To prevent these attacks timestamps have to be located relative to trusted items in the chain.

The problem with this scheme is that we need the cooperation of the other users to produce their timestamps which allow to verify the chain. Alternatively the TSA could store all timestamps and keep them available for everyone but the storage requirement would be prohibitive.

**Extended linking** The protocol can be modified by linking each timestamp with the $k$ previous and next ones ([HS91]). The cooperation of only a subset of these users is required, but the storage space significantly increases.

**Renewing old timestamps** In the event that a hash function used by a timestamping service is broken, all of the timestamps issued prior to that time would be called into question. But if timestamps are renewed with a more secure hash function before the original one is broken, the time evidence contained in the original certificate will still be valid ([BHS92]). It is possible to use two hash functions in parallel so that the system remains secure if one of them is broken.

A NOTE ON INFRASTRUCTURES

Time Stamp Authorities can be incorporated in a public key infrastructure. Certificate Authorities can offer timestamping as an additional feature or the TSA’s can be separate entities. A TSA which also authenticates the clients and verifies the contents of the submitted documents is called a Notary Authority. Entities can be structured in a hierarchical tree and cross-certification between separate trees is possible. In the timestamping scheme described above TSA’s which want to be able to verify each other’s timestamps have to link their chains together periodically.

ACCUMULATING TIMESTAMPS

The previous method which links all timestamps in a linear way poses a very high demand on cooperation and may impose a long computation time before a trusted timestamp is encountered on the chain. The main idea in accumulated hashing is to divide the timestamping procedure into rounds. At the end of each round a timestamp is calculated which depends on all requests submitted during that round and on the timestamp of the previous round. The measure of time now is the length of a round, which will have to be chosen according to the usage of the system and the requirements of the application. In general the stamping procedure works as follows:

1. The client $ID_i$ sends the hash of his document to the TSA: $y_i = h(X_i)$.
2. The TSA accumulates all requests during one round ($y_i$ for $1 \leq i \leq m$) and computes a timestamp based on all these values and on the timestamp of the previous round.
3. The TSA sends to each client the timestamp for the round and data necessary for reconstructing this timestamp. This is the timestamp for his document.

These timestamps can be verified as follows:

1. Compute the hash of the document and check that it is part of the data that reconstructs the timestamp for the round.
2. Check the linking between the timestamp for this round and timestamps for preceding or succeeding rounds (they can be kept available by the TSA) until a trusted value is encountered. This should be a value published in authentic widely available media such as newspapers.

Timestamps for a round can be constructed in an efficient way with one of the following methods:

**Building a binary authentication tree** As explained in [BM91] and [BHS92] all timestamping requests accumulated during a round can be organized into a binary tree structure. The leaves of this tree are the values \(y_i\). They are concatenated by two and hashed to obtain a parent value: \(Z_{12} = H(y_1, y_2)\), etc. At the next level these values (internal nodes of the tree) are again concatenated by two and hashed, for example \(Z_{14} = H(Z_{12}, Z_{34})\). This process continues until we obtain one value, the root of the tree, which is combined and hashed with the timestamp of the previous round to form the timestamp of the current round. When necessary, nodes with random or default values are inserted into the tree so that there is an even number of nodes at each level.

The timestamp of a particular document (represented by a hash value \(y_j\)) from one of the clients consists of the information necessary to rebuild the branch of the tree to which \(y_j\) belongs. This produces the root value and the timestamp of the round. Such a branch contains \(O(\log m)\) values (with \(m\) the number of requests during the round), the computation time is also \(O(\log m)\). This is superior to a straightforward linear hashing of values \(y_i\) where we would require \(O(m)\) operations. The tree is unforgeable if the hash function \(H\) is collision-resistant.

All requests submitted in the same round will have the same temporal measure, but if the TSA builds the tree by applying the values \(y_i\) in a chronological way, the relative positions for the corresponding requests can be preserved. For this however we need unconditional trust in the TSA. An implementation of this method is described in [MAQ99].

**Using a one-way accumulator** In [BM94] a method is proposed where the size of timestamps and computation time are constant with respect to the number of requests \(m\). For this we need a primitive called one-way accumulator. It is a kind of hash function that accumulates an arbitrary number of values in such a way that the order in which these values are processed is irrelevant. The function must also be impossible to invert. This is achieved by the modular exponentiation operation.

The timestamp for a round is constructed starting from a random or default value \(Z_0\) as follows: \(Z = Z_0 \prod_{i=1}^{m} y_i \mod N\). The timestamp for the document

corresponding to the hash $y_j$ consists of the values $Z_j = \prod_{i=1}^{m} y_i \mod N$ and $y_j$. The timestamp for the round is reconstructed by the operation $Z = Z_j^{y_j} \mod N$. (The timestamps are also linked with each other by combining and hashing with the timestamp of the previous round.)

To produce a fake timestamp for a value $y'$ one would have to find a value $Z'$ so that $Z'^{y'} = Z \mod N$. If $N$ is chosen as the product of two distinct primes this is equivalent to solving the RSA problem (which is believed to be as difficult as integer factorization). The modulus has to be chosen large enough (for example 1024 bits), additionally it is suggested in [BM94] to construct it as a rigid integer in order to prevent the reduction of the size of the image after repeated applications of the function.

In this scheme it is no longer possible to preserve the relative positions of the requests submitted in the same round (they are simultaneous). Another drawback is that modular exponentiations are slower than hash functions.

Also note that the accumulator function has a trapdoor, namely the factorization of $N$. The trapdoor function has to be provided by a trusted third party (which can be off-line). In [Nyb96] another accumulator function is proposed which has no trapdoor. However the size of the resulting timestamps seems to be prohibitively large. An analysis of accumulators is made in [Mas98].

**Binary linking schemes** The most recent scheme for timestamping was proposed in [BLLV98]. It is a mixture of the linear linking scheme and the accumulation method using trees. Each timestamp is linked with the previous one and also with another suitably chosen timestamp. Rounds are constructed with a fixed number of requests $m$ (rather than a fixed time slot). The main advantage of the scheme is that the relative position of two arbitrary requests can be determined (without trusting the TSA) even if they were issued in the same round. The verification time and size of the timestamps are $O(\log m)$.

**CONCLUSION**

The interaction between time and cryptography is a recent research subject. We have stressed the importance of time in non-repudiation services and more specific applications. We then described basic mechanisms for providing temporal authentication of electronic documents and discussed the advantages and limitations of several state-of-the-art techniques. Since standardization in this area (by

organizations such as the IETF) is still in early stages further developments and improvements in these systems can be expected.

REFERENCES


