UCL
Université
catholique
de Louvain

TIMESEC

Digital Timestamping and the Evaluation of Security Primitives

Time and cryptography

H. Massias massias@dice.ucl.ac.be
J.-J. Quisquater jjq@dice.ucl.ac.be

March 1997

WP1, Technical Report

Contents

Overview

The interaction between time and cryptography is a new subject. Nevertheless, the time in cryptography is of high importance especially for electronic contracts.

First we must define what we call *time* in cryptography and the different situations we will deal with. Two of the needs are to timestamp a document to answer the question "When has this document been done?" and to send information in the future.

Some solutions were given in the past but they leave some open problems and none have been standardized. There are two kinds of solutions, the one which need a T.T.P. (Trust Third Party) and the others. A solution to date safely is to link the requests. We will see two solutions based on this idea.

Several protocols have been patented, we will study these US-patents. Then we will see applications of the solutions we commented.

None of the protocols which are described in part (1, 3, 4.1) are new. Nevertheless we have precisely describe the verification protocols and we have made the cryptanalyze of all the algorithms. We have studied in details all the aspect of the protocols and then drawn conclusions.

To finish we will proposed a terminology and an informal method based on this work.

Introduction

We will study two needs of dealing with time in cryptography. First the timestamp and then the "timed-release crypto".

Definition 1 *The* **time-stamp** *of a document is something added to it which proves that the document have been issued* **before**, **after** *or* **at** *a certain time.*

Timestamping applies to two main problems in cryptography. The first one is to be able to date securely a digital document and the second one is to extend the lifetime of signatures.

Definition 2 *The goal of* **timed-release crypto** *is to encrypt a message so that it can not be decrypted by anyone, until a pre-determined amount of time has passed.*

It consist on "sending information into the future."

1 Timestamping

The reference for timestamping today is the work of Stuart Haber and W.Scott Stornetta ([1]).

1.1 Several situations

There are several situations in which a document can be timestamped (Figure 1).

In the first situation we make the assumption that we timestamp documents that have just been issued, with the actual time and date.

In the second situation, we timestamp today a document that has been issued in the past, with the time when it was made.

We will only consider the first situation. So no matter when the document was issued, we only worry about when it is submitted to be timestamped.

In this situation, it's **easy** to prove that a document was made **after** a given time, but **difficult** to prove that it was made **before** a given time.
To prove that a document was made after a given time, a solution is to include in the timestamp a widely witnessed event that everybody can date.

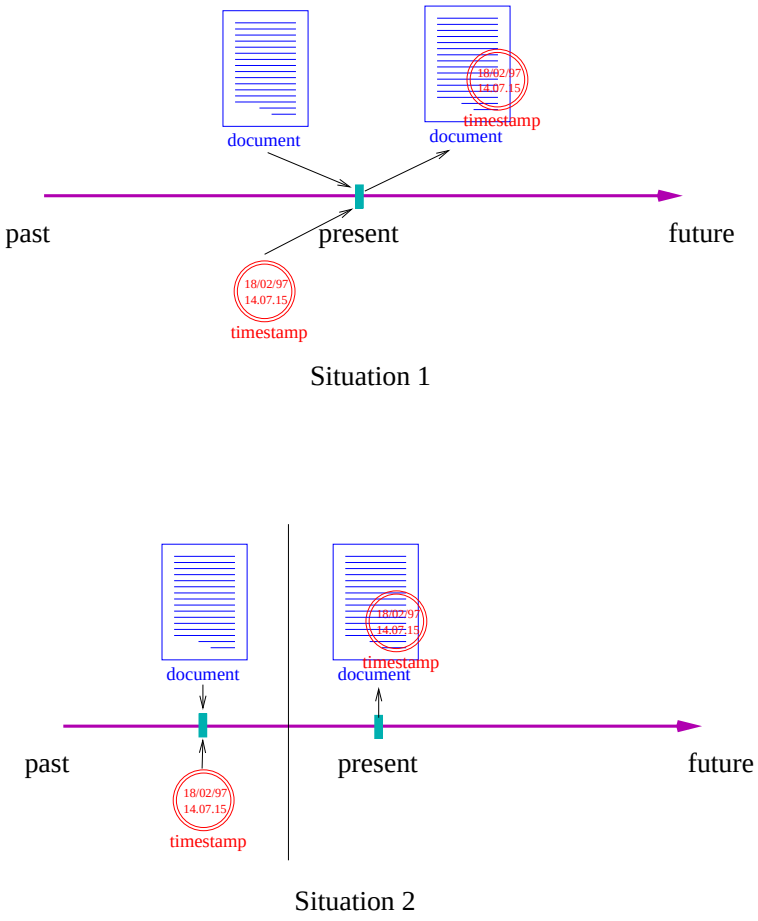

Figure 1: The different situations

1.2 Several solutions

There are two families of protocols.

- The distributed protocols which work without TTP.
- The linking protocols which require a TTP or only a third party.

In all cases, we will date a hash of the document to be timestamped in order to keep the confidentiality of the original document without losing security.

1.3 Distributed protocol

The protocol Alice wants to date a document y, here's the protocol she uses ([1], [3], [5], [7]):

1. First, by using a pseudo-random generator and by initializing it with y, she computes k values $V_1, V_2, \cdots, V_k$.
2. She interprets this k values as identification of k persons, and sends them y.
3. Each of these persons adds date and time to the document before signing it and sending it back to Alice.
4. Alice keeps these k signatures as timestamp of her document.

Explanations To guarantee the safety of this protocol, we have to assume that Alice cannot corrupt k people, so we must choose k big enough.

Initializing the generator with the document y enables the people who wants to check the timestamp to be sure that Alice didn't choose the identity of the k people. We can initialize the pseudo-random generator with a random number that we add in the timestamp (for the verification) but it didn't bring something interesting. We can also imagine that Alice could choose this number in order to obtain identities of people that she can corrupt. A similar idea, but with the protocol we have just defined, is that Alice could make minor changes to his document in order to "choose" the k numbers generated. This must be study for an implementation because it's linked to k, to the hash function used by Alice and to the pseudo-random generator.

The first major drawback of this protocol is the need of enough people able to answer immediately to Alice's request.

The second drawback is the *lifetime of the signatures* which are send back to Alice. This problem will appear each time we use signatures. For this reason Burt Kalisky says that a timestamping protocol must not rely on any secret information.

1.4 Linking protocol

We will denote the TTP by TSS (Time Stamping Service).

1.4.1 Method using chain

A natural solution when we take advantage of a TTP is the following.

We suppose that Alice wants to timestamp y.

1. Alice sends y to the TSS.
2. The TSS adds the date and time, signs and sends it back to Alice.

If the TSS is honest, the only problem is once again the lifetime of the signature.

Let's now improve this protocol in order to reduce the risk of collusion between Alice and the TSS.

To do this a solution consists on linking Alice's document with the documents that the TSS signed before and after, to built an unmodifiable chronological chain. Like this we can be sure that Alice's document was dated before the one which follows and after the one which precedes.

Protocol: ([1]) Let S_k be the TSS signing function, and H a hash function. We suppose that Alice wants to timestamp y_n. It's the n^{th} request made to the TSS.

1. Alice sends y_n and ID_n (her identity) to the TSS.
2. The TSS sends back to Alice:

$$s = S_k(n, t_n, ID_n, y_n; L_n)$$

where t_n is the date and time, and

$$L_n = (t_{n-1}, ID_{n-1}, y_{n-1}, H(L_{n-1}))$$

3. When the next request will be issued, the TSS will send ID_{n+1} to Alice.

(s, ID_{n+1}) is the timestamp for y_n.

L_n is the linking information.

Verification

- First you check $s = S_k(n, t_n, ID_n, y_n; L_n)$.
- Then you ask ID_{n+1} for his timestamp $(n+1, t_{n+1}, ID_{n+1}, y_{n+1}; L_{n+1})$, $L_{n+1} = (t_n, ID_n, y_n, H(L_n))$ and you check if it works.
- You ask ID_{n-1} for his timestamp and see one more time if it works.

Now you can do the same with ID_{n+2}, ID_{n-2} and so on.

Reliability

If we assume that the signature scheme used by the TSS is reliable and that his secret key is still secret, attacks against this scheme can only be made with the collaboration of the TSS.

Let's see the different attacks and what they require.

- We are unable to insert a timestamp in the chain because each message is numbered.
- To make a false timestamp we can replace a document in the chain. To do that we must collude with the TSS and with the people whose requests precedes and follows the one we want to replace. Then we must find a collision in the hash function.

  Let's see in details what it is like.

  We suppose that we want to replace the timestamp corresponding to the n^{th} request. We assume that we want to change the identity and the hashing, i.e. we want to replace $(n, t_n, ID_n, y_n; L_n)$ with $(n, t_n, ID'_n, y'_n; L_n)$.

  - We must have the collusion of ID_{n-1} to replace (s, ID_n) with (s, ID'_n).
  - Then we must have the collusion of ID_{n+1} too. He must replace $L_{n+1} = (t_n, ID_n, y_n, H(L_n))$ with $L'_{n+1} = (t_n, ID'_n, y'_n, H(L_n))$ determined in such a way that $H(L'_{n+1}) = H(L_{n+1})$ in order to be coherent with the timestamp of the $(n+2)^{\text{th}}$ demand.

  If $ID'_n = ID_n$ then the first condition is no more necessary.

  If H is a "good" hash function then the last condition is unworkable.

- Another method to brake the scheme with the collusion of the TSS is to construct a false chain of timestamps long enough to exhaust the most suspicious challenger. To guard against that we can imagine that the TSS publish $H(L_m)$ every day if m is his last request, in order to prevent the modification of $H(L_m)$).

Remarks

A drawback is that everyone must keep all his timestamps to enable the check of the other timestamps. A solution to this is to link the document not only with the preceding and the following one but with the k's.

1. The protocol is the same than the last one except that the linking information becomes

$$L_n = [(t_{n-k}, ID_{n-k}, y_{n-k}, H(L_{n-k})), \cdots,$$

$$(t_{n-1}, ID_{n-1}, y_{n-1}, H(L_{n-1}))].$$

2. When the k next requests have been proceeded, the TSS sends $(ID_{n+1}, \cdots, ID_{n+k})$ to Alice.

A new drawback is that the size of the timestamp grows up, but in compensation it increases the security. To make the same attack that we described before we must not only corrupt ID_{n-1} and ID_{n+1} but $ID_{n-k}, \cdots, ID_{n-1}, ID_{n+1}, \cdots, ID_{n+k}$ too.

More explanation

- Why do we impose to hash the linking information?

  Let's assume that we include L_{n-1} instead of $H(L_{n-1})$ in L_n. Then we include L_{n-2} because it's in L_{n-1} and so on. After each new request the timestamps's size increase, so that becomes quickly unworkable.

  If we decide to link only with the last request $L_n = (t_{n-1}, ID_{n-1}, y_{n-1})$ in order to avoid the use of a hash function, the attack we described earlier is much simple. Now it only requires to be able to modify two requests spaced out one apart. It's no more necessary to find a collision in the hash function.

- In the information L_n that we add in the timestamp, $H(L_{n-1})$ is useful to make the timestamp number n dependent on the preceding ones and to keep a constant size at the same time. $t_{n-1}, ID_{n-1}, y_{n-1}$ are useful for the checking.

1.4.2 Method using tree structure

This scheme uses hash function but **no** keys, i.e. no secret information.

An idea of the protocol is briefly given in ([2]) and an implementation of it is describe by Surety Technologies (see paragraph 4.2.2). We have use that to deduce the protocol which follows.

Protocol (Figure 2)

Let H be a hash function.

We suppose that the TSS receives n requests in the same time unit. The TSS calculates the value HV_i by hashing the documents via a tree structure. The last value HV_i is widely published so that nobody can change it, but everybody can easily get it.

In Figure 2 the y_X, $X = A, B, \cdots, H$ are the documents that the users want to timestamp.

Alice's timestamp contains the information necessary to rebuild the tree branch his document belongs to. The timestamp for y_B is (y_A, right), (H_2, left), (H_6, left).

Verification

The checking simply consists on rebuilding the tree branch and comparing the value that we obtain with the one published for the time unit t_i: HV_i.

Reliability

An attack could be to construct a false tree branch starting with the document we want to timestamp and finishing with the value HV_i corresponding to the time we want to date with.

The assumption that we make is that the tree branch can only be rebuilt with the original values.

We must make sure that this representation doesn't make it easier to find attacks against hash functions.

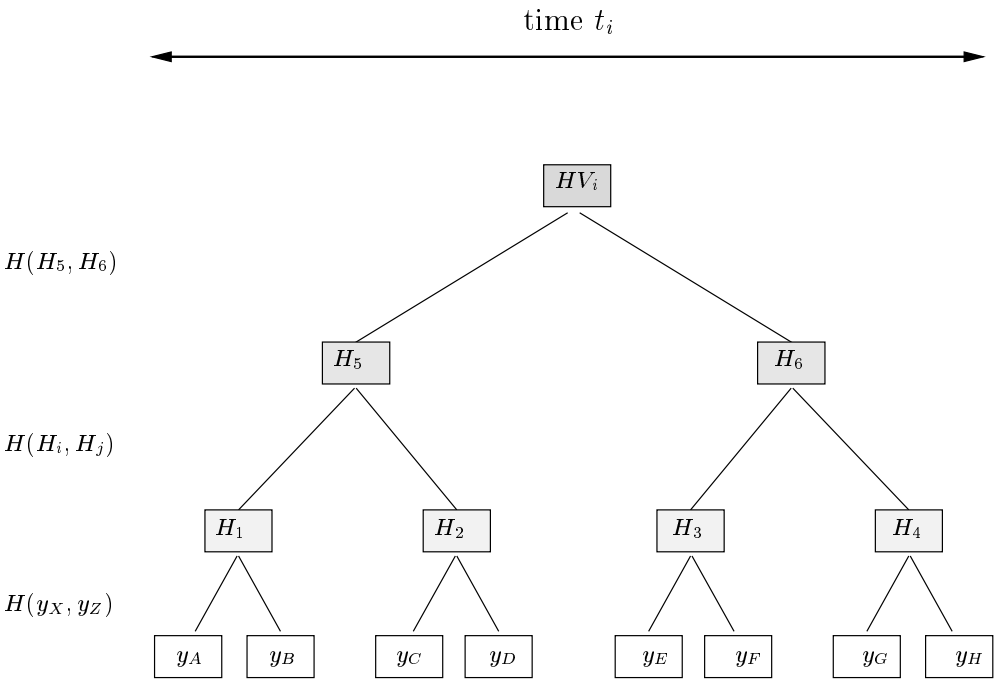

Figure 2: Tree structure

Comments

An advantage is that the checking doesn't depend on the stocking of their timestamps by the other users as opposed to the chaining protocol.

A drawback is that we must have enough requests for each time unit. This must be studied before an implementation.

An improvement

A variation of this protocol consists on linking the values HV_i together with a hash function (Figure 3). We consider the value SHV_{i-1} (Super Hash Value) which is a hashing of all the last HV_j values and we compute $SHV_i = H(SHV_{i-1}, HV_i)$.

An attack

Assume that Charlie wants to make an attack. We suppose that he can't publish the SHV instead of the TSS.

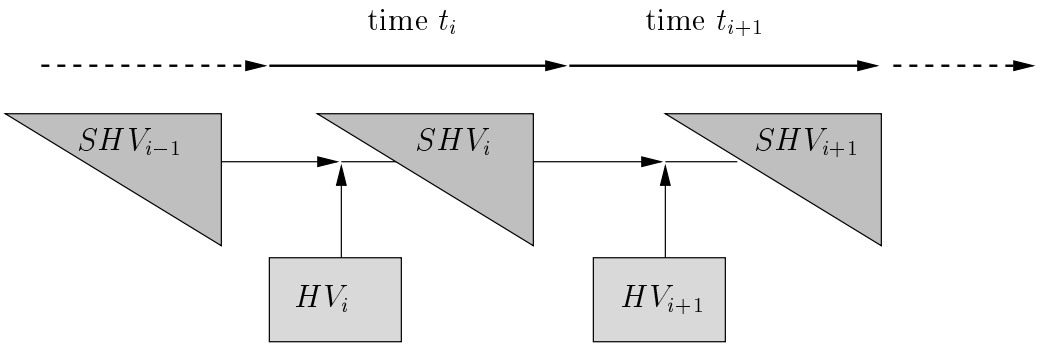

Figure 3: An improvement

Charlie intercepts all the documents to be timestamped and takes the place of the TSS. He will invalidate all the timestamps of the users. If there's no authentication protocol between the TSS and the users it will be very easy and the users will discover deceit only when the SHV will be published. A way to prevent this is that the TSS signs the timestamps. The attack which consist on building a false branch tree will thus become more difficult because it requires the collusion of the TSS.

1.5 The Network Time Protocol: NTP

The introduction of the NTP had enable the computers connected to Internet to know the exact time. The protocol is briefly explain in ([11]). A study of the security has been done by M. Bishop ([12]).

Its only use is to become the exact time, but it didn't enables to timestamp at all.

Explanations and references can be found on the Internet at *http://www.belnet.be/services/ntp.html*.

2 Patents

For the moment four US-patents concerning timestamping protocols have been register. All are from Stuart A. Haber and Wakefield S. Stornetta.

Filed	Date	N°	Title
Aug. 2, 1990	Aug. 4, 1992	5,136,647	Method for secure timestamping of digital documents
Mar. 8, 1991	Aug. 4, 1992	5,136,646	Digital document time-stamping with catenate certificate
Dec. 21, 1992	Dec. 13, 1994	5,373,561	Method of extending the validity of a cryptographic certificate
Nov. 22, 1993	May 30, 1995	RE. 34,954	Method for secure timestamping of digital documents

2.1 US-patent n° 5,136,646

H is a hash function.

1. Alice sends y_n and ID_n (her identity) to the TSS.
2. The TSS sends back to Alice:

$$(n, t_n, ID_n; C_n)$$

where t_n is the date and time, and

$$C_n = H(n, y_n, ID_n, t_n; C_{n-1})$$

$(n, t_n, ID_n; C_n)$ **is the timestamp for** y_n**.**

2.2 US-patent n° 5,136,647

Let s be the TSS signing function.

1. Alice sends y_n and ID_n (her identity) to the TSS.
2. The TSS sends back to Alice:

$$s(C_n)$$

where

$$C_n = (n_{|8}, y_{n|8}, ID_{n|8}, t_{n|8}; y_{n-1|8}, ID_{n-1|8}, t_{n-1|8})$$

$n_{|8}$ means that we truncate n to his last 8 bits.

A big mistake here is to use truncation as a hash function, because it's a very weak hash function.

Another drawback is that the document is only linked with the last request and not with all the preceding ones.

In this form, this patent is useless.

2.3 US-patent n° 5,373,561

The method just consists on timestamping the document with its certificate like we will see in (4).

2.4 US-patent n° Re.34,954

This patent is just the patent n° 5,136,647 which has been reissued with some more claims.

3 Timed release crypto

This problem was first discussed by Timothy May ([9]). Some solutions have been given by Rivest, Shamir and Wagner in ([10]).

Let's now see just the principle of what can be done when we take advantage of a TTP. A good and precise protocol is explained in ([10]).

1. Alice encrypts a message $M' = E(M, K)$ with the key K and wants that M' can not be decrypted before a time t_i.

2. She encrypts K with the public key e_{t_i} and publishes the encrypted value.

3. At time t_i the TTP publishes d_{t_i} which is the secret key corresponding to e_{t_i}.

We believe that this feature can easily be added to a timestamping service.

4 Applications

4.1 Current technical solutions

As we said in the introduction, timestamping has two main applications. The first one is to answer the question: *"When was this document issued? "*.

The second one is to extend lifetime of certain sort of certificates when the protocol used is compromised.

Let's take the example of signatures. Today signatures have a lifetime, it's the time before the secret key is compromised. After that all the signatures that have been made with this key are called into question. That's really a big problem. We can also imagine that someone which wants to repudiate a signature he made, can dishonestly report the compromise of his private key. Then all the signatures he made are invalidated. By doing so any user is able to disavow a signature he regrets.

The protocol (see [2], [6])

Let be D the document and s its signature.

We just timestamp (D, s) and then we are as sure of the validity of the signature as we were when it was timestamped.

In the same way, we can renew timestamps by re-timestamping the document and its first timestamp.

4.2 Implementations

We only know three implementations. Two of them use PGP and work nearly in the same way.

4.2.1 PGP Digital Timestamping services

These services are accessible by email, the explanations are available on the web: *http://www.itconsult.co.uk/stamper.htm, http://www.eskimo.com/ weidai/timestamp.html*.

Here's a brief description of the first service.

Each signature made has a serial number.

- The TSS signs the message.

- He stores all the signatures that he makes. All the detached signatures(serial number, time and date) that have been issued can be inspected.
- Every day the TSS generates two files, one showing the last signature serial number made, and the other containing all the detached signatures made on that day.

  A signature of the files is published every week in a newsgroup.

4.2.2 Digital Notary

It's a product of Surety Technologies (*http://www.surety.com*).

This implementation uses the tree structure (1.4.2).

The time unit is one second. A unique serial number is attached to each document.

The protocol

1. The customer has a software on his personal computer and is connected to the TSS via Internet.
2. Using this software he hashes the original document in a 288 bits hash. He has the option to treat this hash digest singly, or combine it with hash digests from other documents (an "aggregated hash") before transmitting to the TSS.
3. The TSS combine this document(s) with the others that arrived at the same second by hashing them via a tree structure.
4. Each second the TSS construct the SHV. It's published in several places accessible via the network and on CD-ROM too. A value is published every week on the Sunday New-York Times.
5. The TSS sends the customer all that he needs to certify his timestamp: the hash of each document, the computation path followed when all hashes and aggregated hashes were mathematically combined, and the time and date of the HV.

5 Proposed terminology and informal method

We will call STA (Secure Time Authority) the agency which will make secure the use of time for digital documents.

For us *timestamping* is the operation which consist on building the proof that a document was submitted to the STA on a given date.
What we call *timed-release crypto* is the action of keeping an information secret before a given time.

In *timestamping* and *timed-release crypto* there are two global concept. The STA can make the global certification (that we call "global service") or it can give the user the material to do the certification ("self service").

5.1 Global service

A global service for timestamping will consist for the STA on timestamping a document and linking it with other to improve security (see 1.4).

The solution we will adopt is to add the document, the time, the date and some linking information which comes from all the documents that have already been timestamped. At last the STA signs the result before sending it to the user.

A global service for *timed release crypto* would be for the STA to keep the document secret until the expiration date. An alternative would be for the STA to encrypt the document and to keep secret both the document and the key or only the key.

5.2 Self service

We could imagine a self service for timestamping as follow. The STA distributes certified time and date to the user who adds this as a timestamp to his document. The problem is that it just proves that the document has been timestamped after the time contained in the timestamp because the user can keep the timestamp and use it after, or can copy it.

We can prevent from copying the timestamp in the same way as in electronic money. To prevent from using the timestamp later we should force the user to prove that he used the timestamp as soon as he got it.
We can find a way to prove that the timestamp has been used but the problem is to date the proof.

If it's an "on-line" protocol, a solution could be to give a lapse of time beginning when the timestamp is given by the STA. After this lapse of time, if the STA has not receive the proof, the timestamp is invalidated.

A self service for *timed release crypto* works on the idea given in 3.

The STA just produces keys: a public one and the corresponding secret one, and propose a protocol. The user uses the protocol with the public key to encrypt his document. He can now publish the encrypted text which will only be understandable when the STA will publish the secret key.

5.3 Our choice

We will use the NTP to check the STA clock.
We propose to use the global service for timestamping and the self service for timed release crypto because they seems to us the most feasible and they answer the best to the requirements of the project.

References

[1] S. Haber and W.-S. Stornetta, *"How to Time-Stamp a Digital Document"*, Journal of Cryptology, v.3,n.2,1991,pp. 99-112.

[2] D. Bayer, S. Haber and W.-S. Stornetta, *"Improving the Efficiency and Reliability of Digital Time-Stamping"*, Sequences'91: Methods in Communication, Security, and Computer Science; Springer Verlag, 1992, pp. 329-334.

[3] B. Schneier *Applied Cryptography (second edition)*, John Wiley & Sons, inc , 1996.

[4] S. Haber and W.-S. Stornetta, *"Digital document time-stamping with catenate certificate"*, US-patent n° 5,136,646; 1992.

[5] S. Haber and W.-S. Stornetta, *"Method for secure timestamping of digital documents"*, US-patent n° 5,136,647; 1992.

[6] S. Haber and W.-S. Stornetta, *"Method of extending the validity of a cryptographic certificate"*, US-patent n° 5,373,561; 1994.

[7] S. Haber and W.-S. Stornetta, *"Method for secure timestamping of digital documents"*, US-patent n° RE. 34,954; 1995.

[8] S. Haber, B. Kalisky,W-S. Stornetta *"How do digital Time-Stamps support Digital signatures?"*, Cryptobytes, autumn 1995.

[9] Timothy C. May *"Timed-release crypt"*, February 1993.

[10] R. Rivest, A. Shamir, D. Wagner *"Time-lock puzzles and timed-release crypto"*, March 10, 1996.

[11] *"Belnet Info"*, Numéro de référence, 1996, pp.13.

[12] M. Bishop *"A security Analysis of the NTP Protocol (DRAFT)"*, January 1990.