

Implementation of TSP Protocol

---CMSC 681 Project Report

Youyong Zou

Abstract:

This report talks about a network protocol: Time Stamp Protocol (TSP). Time-stamping is a set of techniques enabling one to ascertain whether an electronic document was created or signed at a certain time. First the report introduces the concept of Time Stamp Protocol (TSP), Time Stamp Services(TSS), Time Stamp Authority (TSA). Then the report gives out a design and implementation of this protocol. Last the report give the testing result and conclusion.

1. Overview

Time-stamping is a set of techniques enabling one to ascertain whether an electronic document was created or signed at (or before) a certain time. In practice, most of the time-stamping systems use a trusted third party called Time-Stamping Authority (TSA). Time stamp is a digital attestation of the TSA that an identified electronic document, has been presented to TSA at a certain time. [7]

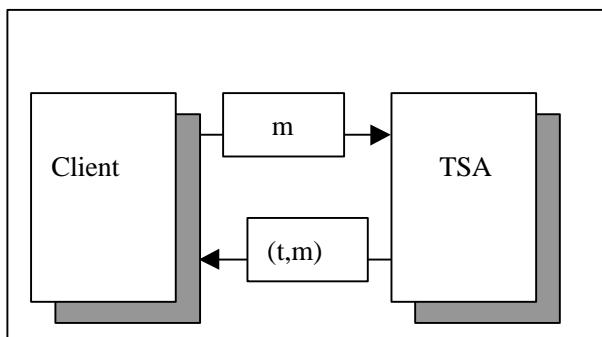


Figure 1: TSP Protocol

The real importance of time-stamping becomes clear when there is a need for a legal use of electronic documents with a long lifetime. Without time-stamping we neither can trust signed documents when the cryptographic primitives used for signing have become unreliable nor solve the cases when the signer himself repudiates the signing, claiming that he has accidentally lost his signature key. During the last years, especially in the context of legal regulation of using digital signatures, the organizational and legal aspects of time-stamping itself have become the subject of world-wide attention. In addition to defining the responsibilities of the owner of the signature, duties and responsibilities of the third party (Time-Stamping Authority, TSA) must be stated as well. Hence, there is an increasing interest in time-stamping systems where the need to trust the TSA is minimized. In order to make users liable only for their own mistakes, there has to be a possibility to ascertain the offender.

I select " Time Stamp Protocol (TSP) in X.509 Public Key Infrastructure " as my project topic.

The draft is in [1]. The paper is divided into three main parts: Background, Solutions and Conclusion. In the first part we define the goals of the project and try to examine in detail the concepts of time-stamping. The following section deals with how I implementation the TSP protocol . Last come the testing and conclusions that we have made regarding the project .

2. Background

2.1 TSP Protocol

Unlike physical objects, digital documents do not comprise the seal of time. Thus, the association of an electronic document uniquely with a certain moment of time is very complicated, if not impossible. Even by the theory of relativity, no absolute time exists. The best we can achieve with time-stamping is the relative temporal authentication based on the complexity-theoretic assumption on the existence of collision-resistant hash functions. Relative temporal authentication enables the verifier given two time-stamped documents to verify which of the two was created earlier.

Time stamping protocol is a ongoing internet draft called " Internet X.509 Public Key Infrastructure Time Stamp Protocols (TSP)" located at . <http://www.ietf.org/internet-drafts/draft-ietf-pkix-time-stamp-11.txt> .

2.2 TSS

A digital time-stamping service (TSS) issues time-stamps which associate a date and time with a digital document in a cryptographically strong way. The digital time-stamp can be used at a later date to prove that an electronic document existed at the time stated on its time-stamp. For example, a physicist who has a brilliant idea can write about it with a word processor and have the document time-stamped. The time-stamp and document together can later prove that the scientist deserves the Nobel Prize, even though an arch rival may have been the first to publish. [6]

Here's one way such a system could work. Suppose Alice signs a document and wants it time-stamped. She computes a message digest of the document using a secure hash function and then sends the message digest (but not the document itself) to the TSS, which sends her in return a digital time-stamp consisting of the message digest, the date and time it was received at the TSS, and the signature of the TSS. Since the message digest does not reveal any information about the content of the document, the TSS cannot eavesdrop on the documents it time-stamps. Later, Alice can present the document and time-stamp together to prove when the document was written. A verifier computes the message digest of the document, makes sure it matches the digest in the time-stamp, and then verifies the signature of the TSS on the time-stamp.

2.3 TSA

The TSA is a Trusted Third Party that creates time stamp tokens in order to indicate that a datum existed at a particular point in time. In order to associate a datum with a particular point in time, a Time Stamp Authority (TSA) may need to be used. This Trusted Third Party provides a "proof-of-existence" for this particular datum at an instant in time. [1]

The TSA's role is to time stamp a datum to establish evidence indicating the time at which

the datum existed. This can then be used, for example, to verify that a digital signature was applied to a message before the corresponding certificate was revoked thus allowing a revoked public key certificate to be used for verifying signatures created prior to the time of revocation. This is an important public key infrastructure operation. The TSA can also be used to indicate the time of submission when a deadline is critical, or to indicate the time of transaction for entries in a log. An exhaustive list of possible uses of a TSA is beyond the scope of this document.

Requirements of the TSA: [1]

- use a trustworthy source of time.
- include a trustworthy time value for each time stamp token.
- include a unique integer for each newly generated time stamp token.
- produce a time stamp token upon receiving a valid request from the requester, when it is possible.
- include within each time stamp token an identifier to uniquely indicate the security policy under which the token was created.
- only time stamp a hash representation of the datum, i.e.a data imprint associated with a one-way collision resistant hash-function uniquely identified by an OID.
- examine the OID of the one-way collision resistant hash-function and to verify that the hash value length is consistent with the hash algorithm.
- not examine the imprint being time stamped in any way (other than to check its length, as specified in the previous bullet).
- not include any identification of the requesting entity in the time stamp tokens.
- sign each time stamp token using a key generated exclusively for this purpose and have this property of the key indicated on the corresponding certificate.
- include additional information in the time stamp token,if asked by the requester using the extensions field, only for the extensions that are supported by the TSA. If this is not possible, the TSA SHALL respond with an error message.

2.4 TSP Transactions

As the first message of this mechanism, the requesting entity requests a time stamp token by sending a request (which is or includes a TimeStampReq, as defined below) to the Time Stamping Authority. As the second message, the Time Stamping Authority responds by sending a response (which is or includes a TimeStampResp, as defined below) to the requesting entity. [1]

Upon receiving the response, the requesting entity shall verify the status error returned in the response and if no error is present it shall verify the various fields contained in the TimeStampToken and the validity of the digital signature of the TimeStampToken. In particular, it shall verifythat what was time stamped corresponds to what was requested to be time stamped. The requester shall verify that the TimeStampToken contains the correct certificate identifier of the TSA, the correct data imprint and the correct hash algorithm OID. It shallthen verify the timeliness of the response by verifying either the time included in the response against a local trusted time reference, if one is available, or the value of the nonce (large random number with a high probability that it is generated by the client only once)

included in the response against the value included in the request. For more details about replay attack detection, see the security considerations section (item 6). If any of the verifications above fails, the TimeStampToken shall be rejected.[1]

Then, since the TSA's certificate may have been revoked, the status of the certificate SHOULD be checked (e.g. by checking the appropriate CRL) to verify that the certificate is still valid.

Then, the client application should check the policy field to determine whether or not the policy under which the token was issued is acceptable for the application.

2.5 Application of TSP

TSP can be used in many security related application areas[5]

- General WWW Security Mechanism
 - IP Address Authentication
 - Basic Authentication
 - install NCSA Server
 - install CERN Server
 - Message Digest Authentication
- PGP / PEM mail
- S-HTTP (Secure-HTTP)
- SSL / SSLeay: Transport Layer Security
 - secure channel
 - any TCP-based protocol: HTTP, NNTP, telnet, ftp, etc
 - optional public key server / client authentication (by using RSA, DSS, X509)
 - hash: combined MD5 & SHA

3. Design

The system includes two parts: client and server. Client will compute message digest number and send it to server. TSA will accept request and put digest number and current time together, then sign it with its certificate, send response to client. Client can verify it with server's certificate.

3.1 Generate Digest Number

TSP is used to time stamp a data such as a file or an email. First step here is to compute a message digest number for this file. The most used digest algorithms include MD5, MD2, MD4, SHA1, SHA, MDC2. These algorithms produce a short fixed-length string of bits, referred to as "digital fingerprints", "hash values" or "message digests", that uniquely represents (is sufficient to identify) a single piece of digital data. Digital fingerprints are unique in the sense that two different pieces of data, when run through this digest algorithms, can never create the same digital fingerprint.

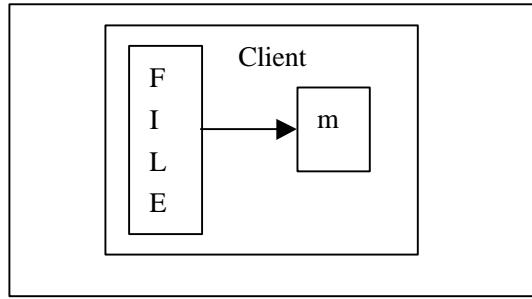


Figure 2: Generate Digest Number

3.2 Send Request to TSA

The client will send message digest number to TSA. The message is packed in a structure called TimeStampReq. It includes following parts:

- Version number;
- Message Digest Algorithm used
- Message Digest Number
- Request Policy
- Nonce
- Certificate Request

TSA is listening on well known TSP request port (standard port is 318, In my implementation, I use a big port number: 8633).

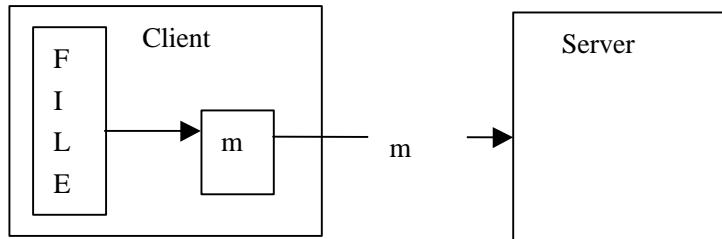


Figure 3: Send Request to TSA

3.3 Sign Digest Number

TSA read request from socket , extract the message digest number. TSA will put it with date and time and generate a time stamp token for client. TSA will use its private key to sign this token.

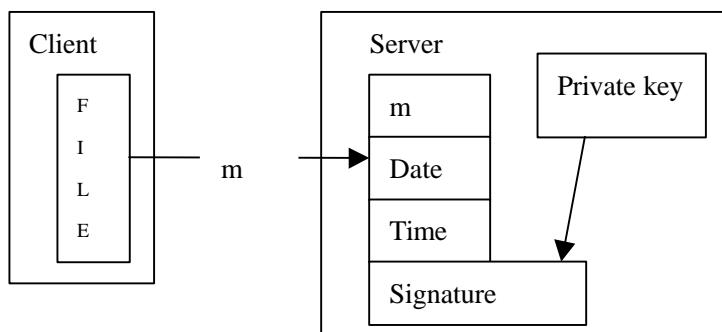


Figure 4: Sign Digest Number

TimeStampToken also include a serial number of this token. It is unique for this TSA. I did not make use of it in my implementation. Actually it is very useful in area like Real Time Protocol (RTP).

3.4 Send Response

The time stamp token will be formatted to base64 format to fit in timeStampResp structure and been sent back to client through socket.

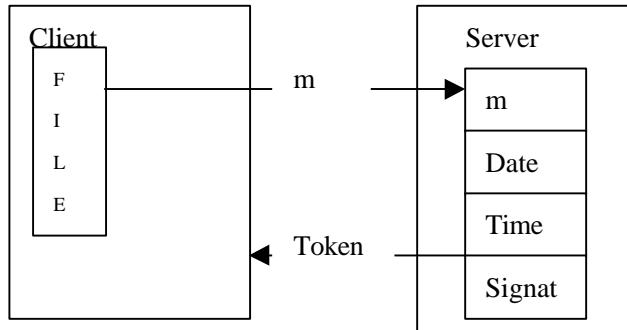


Figure 5: Send Response

TheTimeStampResp includes Status of the request. The status can be “granted”, or if the request is refused, the failure information must be given in FailInfo field.

3.5 Verify

The client or other user which received the time stamped token can verify the token. There are two verification involved. First is the verification of Signature, make sure it is signed by trustable third party. Client can download TSA’s certificate through PKI interface (it may be LDAP or HTTP interface , my implementation use HTTP interface). Because token is signed by TSA’s private, it can be verified by using TSA’s certificate which include a public key matching the TSA’s private key. Second verification is to verify this time stamp is really for this file. Client can decrypt the token, extract the message digest number from it, and compare it with the original message digest number. If it doesn’t match, either time stamp is broken, or time stamp token is not for this file or this version..

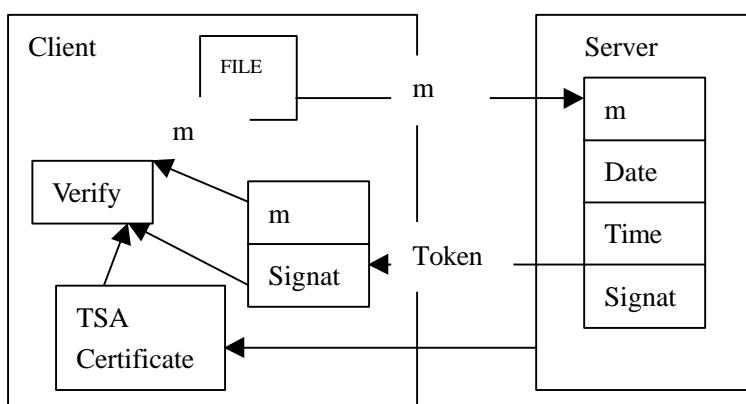


Figure 6: Verify

If the user want to verify the time of timestamp is before a specified date or time, they can extract the date and time field from token and compare with specified time.

4. Implementation

This section will discuss the implementation of TSP protocol. The implementation is made in c and test in SUN Solaris system.

In order to provide the best possibilities to complete the implementation on time I decided to use the ssleay. Ssleay is a open source project which provide security function . Many security products are based on it, such as openssl. I could then implement most of the security related functionality based on it, without having to programming PKI and security.

4.1 Definition

```
/* define of PKI status */
#define GEANTED      0 /* a time stamp token is presented */
#define GRANTEDWITHMODS 1 /* time stamp token with modification is presneted */
#define REJECTION 2
#define WAITING     3
#define REVOCATIONWARNING 4 /* warning that a revocation is imminent */
#define REVOCATIONNOTIFICATION 5 /* notify that a revocation has occured */

/* define of failure reason */
#define BADALG      0 /* unrecognized algorithm ID */
#define BADREQUEST   2 /* transaction not supported */
#define BADDATAFORMAT 5 /* data submited has wrong format */
#define TIMENOTAVAILABLE 14 /* TSA's time source is not available */
#define UNACCEPTEDPOLICY 15 /* requested TSA policy not supported */
#define UNACCEPTEDEXTENSION 16 /* requested extension not supported */
#define ADDINFONOTAVAILABLE 17 /* additional information could not be
understood */
```

4.2 Client

TSP client's function is to accept input from user, connect to TSP server, output reply data.

TSP Request data structure

```
typedef struct {
    int      version; /* default v1(1) */
    char    hashAlgorithm[MAXLEN];
    /* hash function to be used, must be a known algorithms,
       such as : SHA-1 or MD5*/
    char    hashedMessage[80];
    /* a OCTET string, its length must match the length of hash
       value for that algorithm ( eg. 20 byte for SHA-1, 16 for MD5) */
    int      reqPolicy;
    /* TSA policy which the TimeStampToken SHOULD be provided */
```

```

int      nonce;
/* verify the timelines of the response when no
   local clock is available */

int      certreq ;
/* TSA's public key certificate MUST be
   provided if set to 1 */
char     client_name[50];
char     extensions[MAXLEN];
/* additional information for future */

} TimeStampReq;

```

When client send out request , it will set a timeout clock. If get no response after timeout, client will print a error message , stop waiting.

When connected to TSP server, we know where is the TSA's certificate file. Here, certificate is access through http. The client will call wget to get certificate and save it to a file, which can be used to verify time stamp token later.

4.3 Server

The job of TSP server is to listening at TSP port, accept the request, sign the message digest number in the request and send back response.

The structure of TSA response and time stamp token.

```

typedef struct TimeStampResp {
/* PKI Status */

    int      status;      /* PKI status */
    char     statusString[MAXLEN]; /* PKI free text */
    int      failinfo;    /* reason of time stamp rejection */

/* TimeStamp Token */

    char     contenttype[10]; /* id-signedData */
    char     content[256];   /* signedData */

    char     failinfo[MAXLEN]; /* PKI Failure information */

} TimeStampResp;

typedef struct {
    int      version;    /* default v1(1) */
    char     hashAlgorithm[MAXLEN];
    /* hash function to be used, must be a known algorithms,
       such as : SHA-1 or MD5*/
    char     hashedMessage[80];
    /* a OCTET string, its length must match the length of hash
       value for that algorithm ( eg. 20 byte for SHA-1, 16 for MD5 ) */
    int      reqPolicy;

```

```

/* TSA policy which the TimeStampToken SHOULD be provided */
int      serialNumber; /* timestamp serial number */
char     gentime[60];
int      accuracy; /* second, millis or micros */
int      ordering; /*Time stamp token ordered based on gentime */
char     extensions[MAXLEN];
/* additional information for future */
} TimeStampToken;

```

Server will extract digest number from request and put it to TimeStampToken. Also, server will copy other value from request. SerialNumber is a number which keep increasing. Server use ssleay to sign the token and change the signed token to a base64 string. This string will be attached to response message and send back to client.

5 . Test

I do my testing on two sun machines. I use a self-signed certificate generated by openssl as TSA's certificate. It is put on web, so client can get it through http interface.

1). **start tsp_server** yzou1.p12 at a shell window at one machine like sunserver1

sunserver1\$tsp_server yzou1.p12

Start TSP SERVER at : sunserver1

Certificate file: yzou1.p12

Download from <http://www.cs.umbc.edu/~yzou1/yzou1@cs.umbc.edu.p12>

Listening port 8633...

2). **start tsp_client** at another shell window at one machine like grad-sun-06

3). type help in client shell window

Commands are:

HELP - Print helpful information about a command

DIGEST - Compute a Message-Digest number for a file

CONNECT - Connect to TSP server

CERTIFICATE - Get TSP server's certificate, save to file

VERIFY - Verify a Timestamp

REQUEST - Request a time stamp token from a TSP server

QUIT - Exit tsppki console

INFO - Show information

4). type **connect sunserver1** at client shell window

TSP_PKI> connect sunserver1

Connect to TSP Server : sunserver1

The x509 certificate is located at

<http://www.cs.umbc.edu/~yzou1/yzou1@cs.umbc.edu.p12>

5). type **certificate** ServerCertificate ta.cer at client shell window

(copy Server certificate location from output at last step)

TSP_PKI> certificate <http://www.cs.umbc.edu/~yzou1/yzou1@cs.umbc.edu.p12> ta.cer

```
./wget http://www.cs.umbc.edu/~yzou1/yzou1@cs.umbc.edu.p12 -O ta.cer
--11:23:27-- http://www.cs.umbc.edu:80/%7Eyzou1/yzou1%40cs.umbc.edu.p12
=> `ta.cer'
Connecting to www.cs.umbc.edu:80... connected!
HTTP request sent, awaiting response... 200 OK
Length: 3,099 [text/plain]
OK -> ... [100%]
11:23:27 (1.48 MB/s) - `ta.cer' saved [3099/3099]
6). type digest email.txt at client shell window.
TSP_PKI> digest email.txt
./ssleay dgst email.txt
MD5(email.txt)= 9062818909d0088b
7). type request email.ts MD ( MD is the number output at last step)
TSP_PKI> request email.ts 9062818909d0088b
Send TSP request to Server sunserver1
hashAlgorithm = MD5
hashedMessage = 9062818909d0088b
Waiting for response from Server
Got Response from TSA Server
statusString = Granted
contenttype = pkcs
Decrypt the signed TimeStampToken
./ssleay enc -in email.ts -out tmpfile.3 -d -a -k abc
Version = 1
hashAlgorithm = MD5
hashedMessage = 9062818909d0088b
gentime = Wed Dec 31 19:00:00 1969
ordering = 1
accuracy = 1
8). type verify ta.cer email.ts at client shell window
TSP_PKI> verify ta.cer email.ts
./ssleay verify -CAfile ta.cer email.ts
load certificate file
OK
9). type quit at client shell window
```

The length of digest number is different based on different digest algorithms, Such as, SHA-1 is 20 byte, MD5 is 16 byte.

The verify function can only the signature of token with TSA's certificate. User can type **digest** to compute digest number and compare with the output of response from TSA.

6. Conclusion

I have spent many days over this project, but what it all comes down to in the end is I have learned from it. The most important lesson was to see how hard it can be to do thing in practice

that seem simple enough in theory. Naturally, I have learned a great deal about time-stamping and what it can be used for. It was interesting to learn that time-stamping can be used for security problems and e-commerce.

I try to compare TSP solution with x509 certificate when I was doing this project. TSP use PKI to sign document and verify. It drops one of primary concern of PKI: **security**. When message digest number is sent to TSA from client. It is clear text. Others can see it and get it. Then message is send back to client, every one can also see it and use TSA's certificate(it is public) to decrypt it. There is a problem: you can use a time stamped token to verify this paper is published 5 years ago. Others can use forge a time stamped token to verify this paper is published 5 days ago. I was doing a project which make use of certificate's extension field. Maybe this is a solution for time stamp. Client can generate digest number , generate public key/private key pair, send a certificate request to CA (here CA do the function of TSA). This request is encrypted by CA's public key, so no one else can hack it and know the message digest number. CA get the request and generate a certificate for this request, put digest number and time in certificate 's extension field (only X509 Version 3 support extension field), send it back to client. It is easy to verify because it is signed by CA, whose certificate is public.

The use of a TSS would appear to be extremely important, if not essential, for maintaining the validity of documents over many years. In the future, it is likely that a TSS will be used for everything from long-term corporate contracts to personal diaries and letters. Today, if an historian discovers some lost letters of Mark Twain, their authenticity is checked by physical means. But a similar find 100 years from now may consist of an author's computer files; digital time-stamps may be the only way to authenticate the find.

7. References

- [1] "Internet X.509 Public Key Infrastructure Time Stamp Protocols (TSP)", C. Adams, D. Pinkas, Patrick Cain, Robert Zuccherato,
<http://www.ietf.org/internet-drafts/draft-ietf-pkix-time-stamp-11.txt>
- [2] T. Dierks, C. Allen, "The TLS Protocol, Version 1.0," RFC 2246, January 1999.
- [3] C. Adams, S. Farrell, "Internet X.509 Public Key Infrastructure, Certificate Management Protocols," RFC 2510,March 1999.
- [4] R. Housley, W. Ford, W. Polk, D. Solo, "Internet X.509 Public Key Infrastructure, Certificate and CRL Profile," RFC 2459, January 1999.
- [5] "Time Stamp Protocol" <http://www.univ-tln.fr/~byun/timestamp.html>
- [6] Choi, Yong-hwan "Trends of Time Stamp services Technology standard"
<http://www.kisa.or.kr/en-kisa/newsletter/200004/special7.html>
- [7] Digital Signatures and Authentication Time-Stamping
<http://www.cyber.ee/research/publications/auth/index.html#tstamp>
- [8] "Digital IDSM Introduction"
http://digitalid.verisign.com/client/help/id_intro.htm#time_stamp
- [9] Karl-Fredrik Blixt " Adding Non-Repudiation to Web Transactions"
<http://www.it.isy.liu.se/~asa/publications/NonRepudiation.html>