Timestamp Authority

SIL 765 - Assignment 2

Nichit Bodhak Goel (2017MCS2089)
Shadab Zafar (2017MCS2076)

Project Selection:

A1 = Last 4 digits of entry number of first student = 2089
A2 = Last 4 digits of entry number of second student = 2076

K = A1 + A2 mod 3 = 4165 mod 6 = 1

Project 1: Implementation of Timestamp Authority

Problem statement

This application relates to time-stamping a document that one may have prepared some moments ago. The process envisaged is: uploaded the document to server (or some version thereof) and expect to receive the same but with the current date and time stamped onto the document. Thus there is exists a "GMT date & time-stamping server" which has the correct GMT date and time. It uses that to timestamp document (in some standard format) with the current GMT date/time and a digital signature. At any time, it should be possible to establish the fact that the document existed at the date/time stamped, and that the document has not been modified.

Further:

1. How do you ensure privacy, in that the server does not see/have/keep the original document?

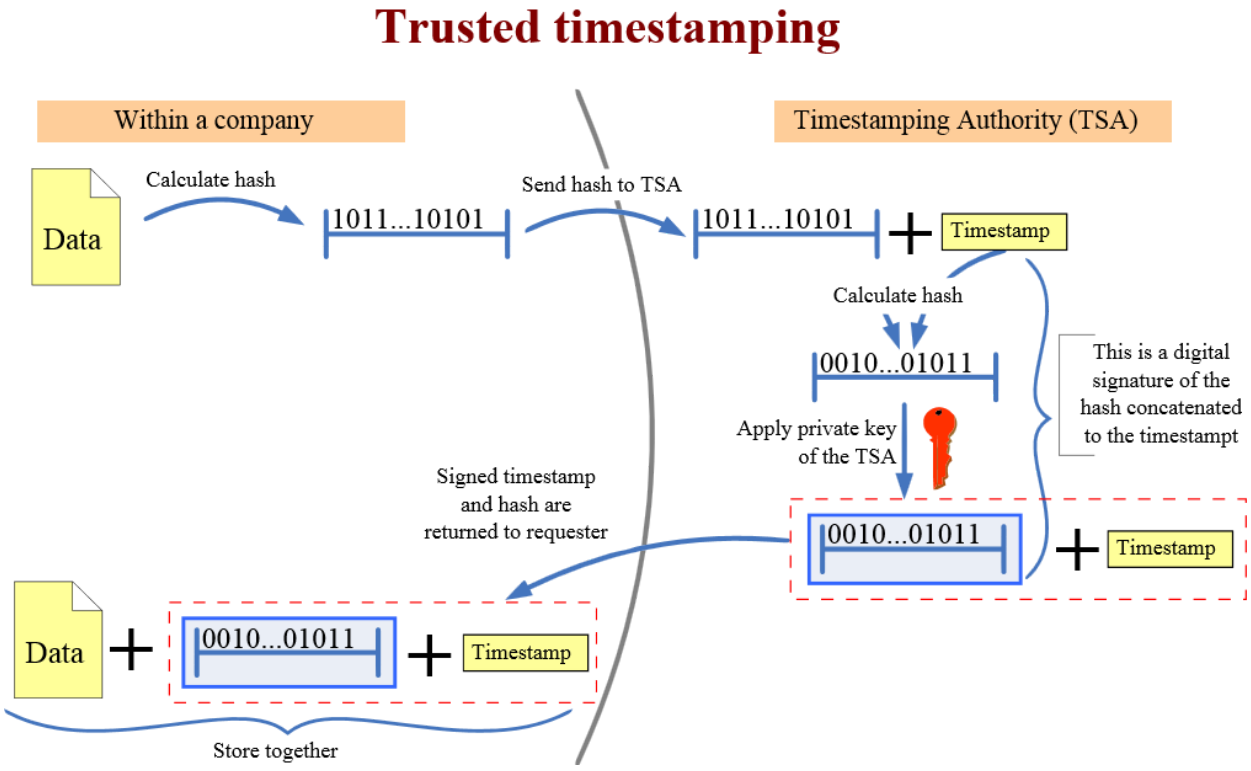
2. How do you share the document with others in a secure manner with the date/time preserved, and integrity un-disturbed?

3. Also ensure that the user has (and uses) the correct "public-key" of the GMT date/time stamping server.

Introduction

Suppose Alice wants to prove to Bob that she had access to a document at time 't' then, she can do so by getting the doc time stamped by a trusted timestamping authority (TSA) server that both Alice and Bob trust. In order to maintain privacy, authentication, integrity and confidentiality - hashing, digital signature and encryption algorithms are used in the implementation.

Creating a timestamp

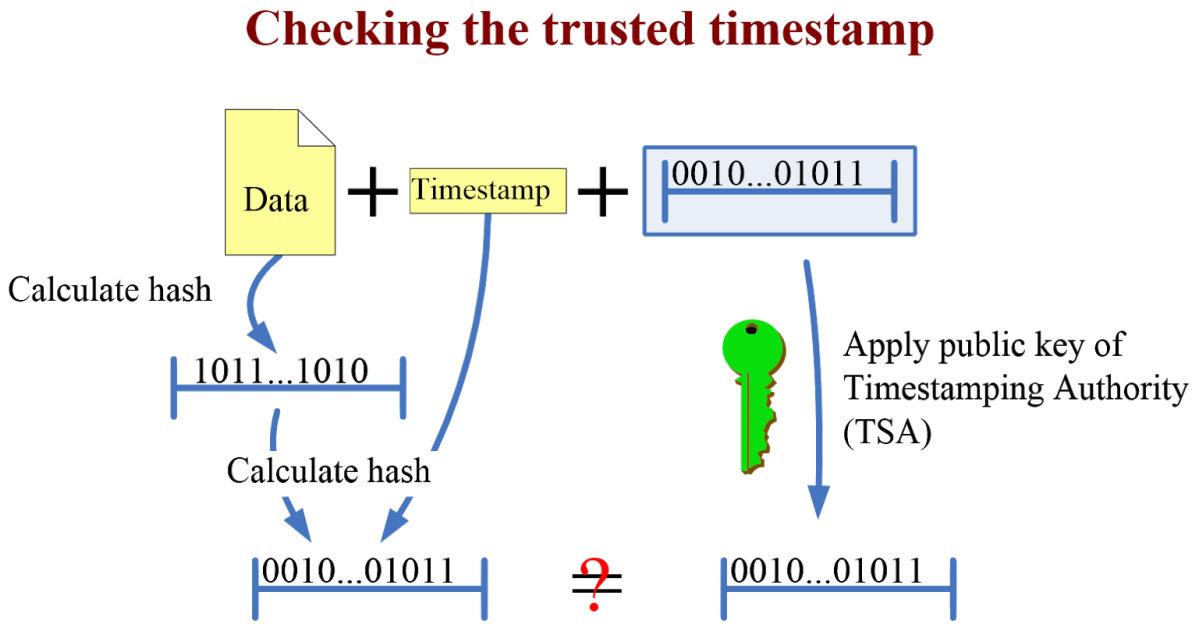


1. A hash is calculated from the data.
2. This hash is then sent to the TSA (timestamp authority).

3. The TSA concatenates the current GMT timestamp to the hash and calculates the hash of this concatenation.
4. This hash is in turn digitally signed with the private key of the TSA.
5. This signature and the timestamp is sent back to the requester of the timestamp who stores these with the original data.

(Data + Timestamp + Signature) can now be sent to anyone requesting the document.

Checking the timestamp



1. Hash (D_H) of the original data is calculated.
2. The given timestamp is concatenated with the data hash (D_H)

3. Hash (A) of the result of this concatenation is calculated
4. The given signature is decrypted using Public Key of TSA, resulting in Hash B.
5. If A & B are equal, then the timestamp and message is unaltered and was issued by the TSA.
6. Otherwise, either of timestamp/document was altered or the timestamp was not issued by the TSA.

Implementation

1)Time Stamping Authority (TSA) Server :-

a. Initially we will start the timestamp authority server since it should be always ready to accept any incoming requests.
b. Also, the TSA can handle multiple requests simultaneously since for each request, it will spawn a new thread which will handle the request of that particular client.
c. Once a request is received, it will timestamp that document's hash and digitally sign it i.e encrypt using RSA algorithm. After that it will return the (timestamp, signature) pair to the client.

```
$ py3 server.py

> Timestamp authority now listening on port: 7171

>>> New client connected: ('127.0.0.1', 49500)
> Received a document's hash:
ddd62c4ffcdd487a8299359c6e680c3696fdffc3942fdb36eb25d3540cff12d3
> Sending timestamp and signature to client
>>> Client disconnected:  ('127.0.0.1', 49500)
```

2) Client - Requester

a. Client will be having an input file (Doc.txt) which needs to be timestamped.
b. Client will calculate the document's hash by using SHA256 and send this hash to the TSA server.
c. After receiving timestamp and digital signature from TSA server, it will encrypt the Doc + timestamp + digital signature using RSA and save it in a file Doc_stamped.txt.
d. This file will be sent to the receiver (client_verifier).

---

3)Client - Receiver / Verifier

a. Receiver will receive the file from the client and decrypt it using his own private key ensuring confidentiality. After decrypting, receiver will get Doc + timestamp + digital signature.
b. Then, receiver will find the hash of Doc(H) and also decrypt the digital signature using TSA server's private key.
c. Now, if Hash of (H + timestamp) = Decrypted digital signature, then integrity is maintained.

```
$ py3 client_verifier.py Doc_stamped.txt

> Given input file: Doc_stamped.txt
> Document hash:
ddd62c4ffcdd487a8299359c6e680c3696fdffc3942fdb36eb25d3540cff12d3

> Document has not been modified and was available at:
2018-03-18 20:02:41.697669
```

1. How do you ensure privacy, in that the server does not see/have/keep the original document?

Ans. To ensure privacy, the document is hashed using SHA256 algorithm before sending it to the server. In this case, the server doesn't know the actual contents of the file and after timestamping, it returns a digitally signed copy of the document hash along with the timestamp and thus the document is not stored in the server.

2. How do you share the document with others in a secure manner with the date/time preserved, and integrity un-disturbed?

Ans. To share the document in a secured manner, we can use symmetric key cryptography ex. AES, DES etc (since they are fast and works on large files) to encrypt the data and share the symmetric key using public key cryptography algorithm for ex. RSA, Elgamal etc. But as the document we used is very small in size so we directly used RSA for encryption of the (document , timestamp and digital signature) with receiver's public key. Then, only the receiver can decrypt the document with its private key. Thus, maintaining the confidentiality of the message.

The signature acquired from the TSA can be used to ensure integrity and that date/time is preserved. (Provided the receiver trusts the TSA)

3. Also ensure that the user has (and uses) the correct "public-key" of the GMT date/time stamping server.

Ans. To ensure that the user uses the correct public-key of the TSA, we need to have a trusted central authority in the system.

This authority could be the TSA itself (In which case we can obtain the public key from TSA and assume it is the correct key), or a different entity that provides a certificate which binds TSA's public

key to TSA’s identity.

References:

1. Trusted Timestamping - Wikipedia
2. RFC 3161