Timestamp Authority

Implementation in Python

2017MCS2089 - Nichit Bodhak Goel
2017MCS2076 - Shadab Zafar

Introduction

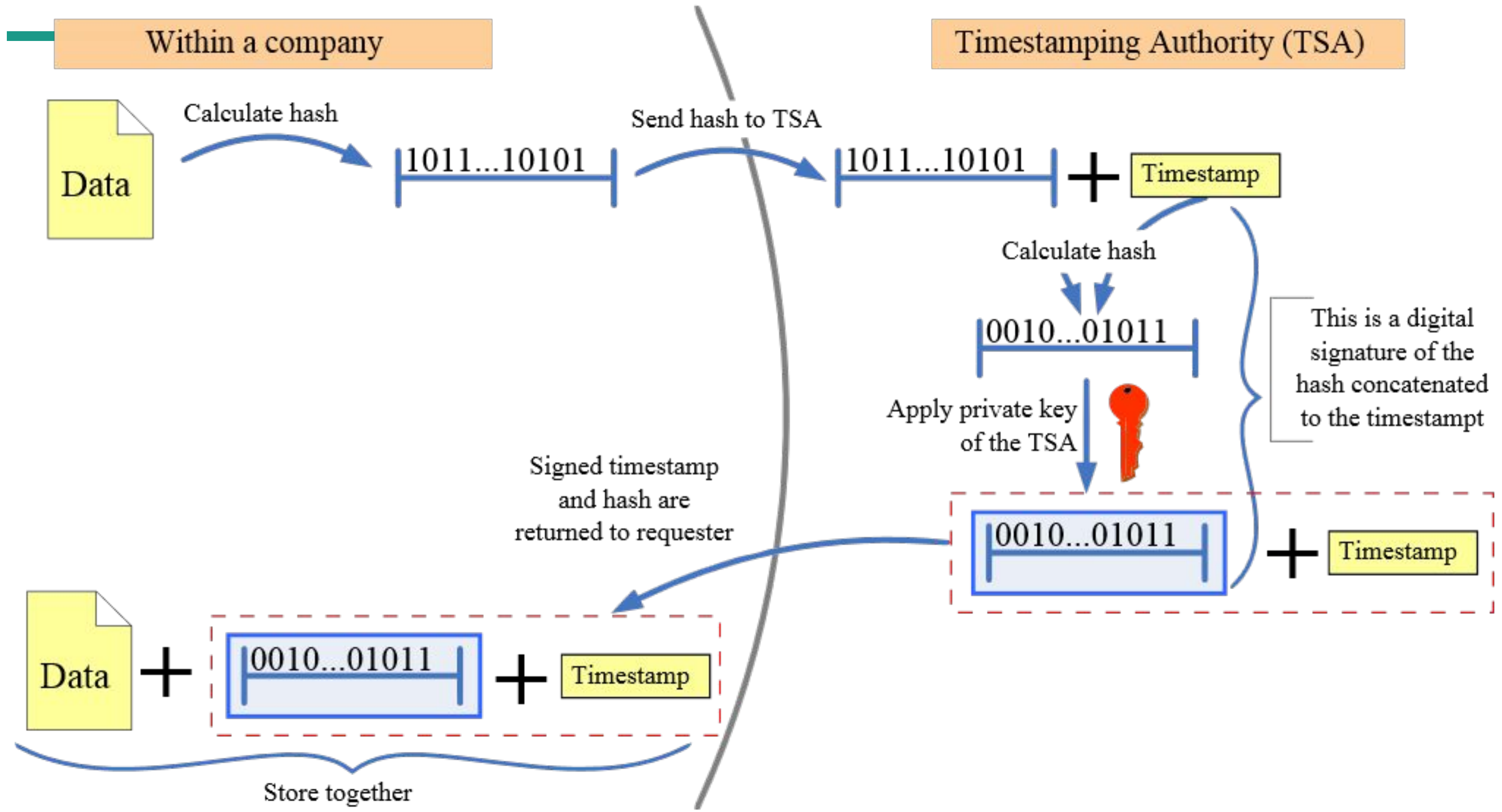
- Suppose Alice wants to prove to Bob that she had access to a document at time ‘t’ then, she can do so by getting the doc time stamped by a timestamping authority (TSA) server that both Alice and Bob trust.

- We used:
    - Hashing to prevent the TSA from viewing the document and ensuring integrity during transmission.
    - Digital Signatures for ensuring authentic timestamps.
    - RSA encryption for ensuring confidentiality during transmission.

Procedure for Creation of 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.
6. (Data + Timestamp + Signature) can now be sent to anyone requesting the document.

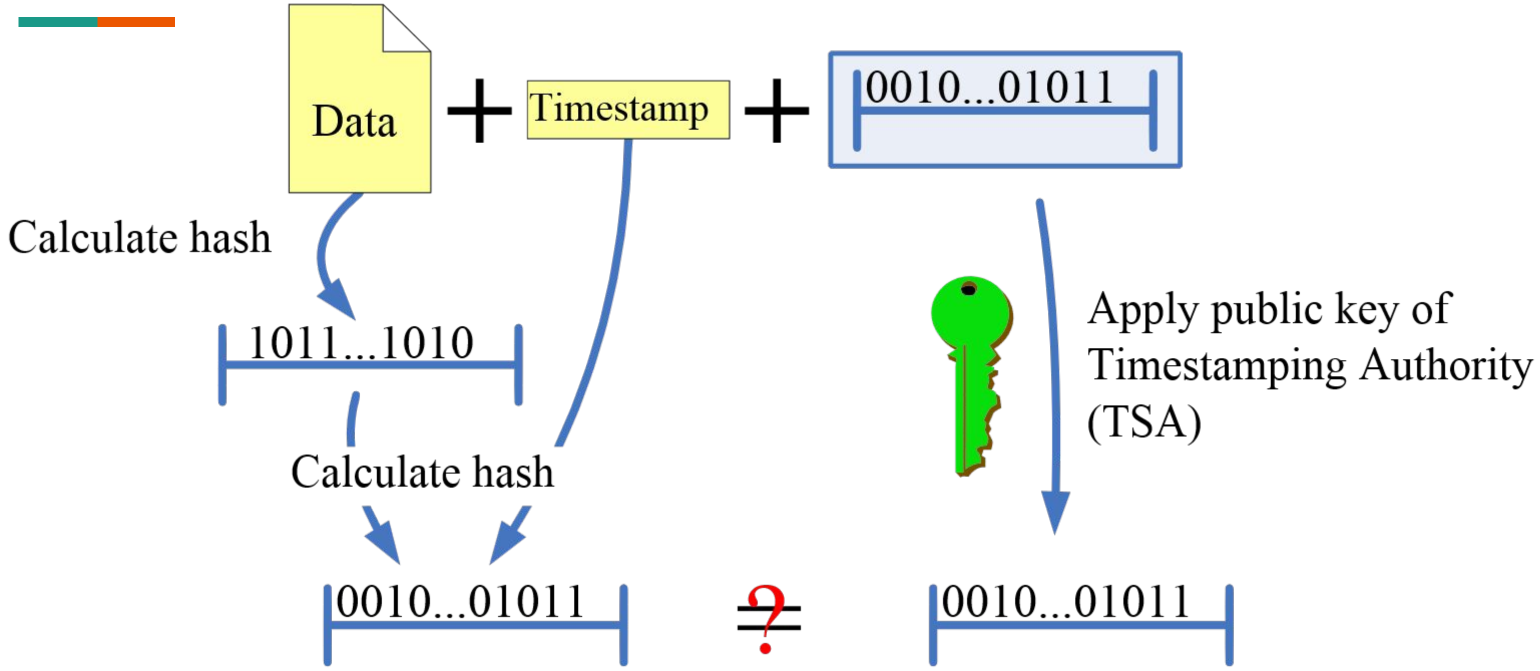
Trusted timestamping


Within a company
Timestamping Authority (TSA)
Data
Calculate hash
1011...10101
Send hash to TSA
1011...10101
Timestamp
Calculate hash
0010...01011
This is a digital signature of the hash concatenated to the timestampt
Apply private key of the TSA
0010...01011
Timestamp
Signed timestamp and hash are returned to requester
Data
0010...01011
Timestamp
Store together

Procedure for 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.

Checking the trusted timestamp



If the calculated hashcode equals the result of the decrypted signature, neither the document or the timestamp was changed and the timestamp was issued by the TTP. If not, either of the previous statements is not true.

Output

Time Stamping Authority

```
$ py3 server.py

> Timestamp authority now listening on port: 7171

>>> New client connected: ('127.0.0.1', 49500)
> Received a document's hash:
ddd62c4ffcdd487a8299359c6e680c3696fdffc3942fdb36eb25d3540cff12d
3
> Sending timestamp and signature to client
>>> Client disconnected:  ('127.0.0.1', 49500)
```

Output

Client (Alice)

```
$ py3 client_requester.py Doc.txt

> Given input file: Doc.txt
> Sending document hash to server:
ddd62c4ffcdd487a8299359c6e680c3696fdffc3942fdb36eb25d3540cf
f12d3

> Received timestamp & signature from server
> Dumping data to: Doc_stamped.txt
```

Output

Receiver (Bob)

```
$ py3 client_verifier.py Doc_stamped.txt

> Given input file: Doc_stamped.txt
> Document hash:
ddd62c4ffcdd487a8299359c6e680c3696fdffc3942fdb36eb25d3540cf
f12d3

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

Questions

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.

THANK YOU