A human-readable summary of the X.509 PKI Time-Stamp Protocol (TSP)

Daan Sprenkels

Radboud University Nijmegen, The Netherlands
dsprenkels@science.ru.nl

1 Introduction

In August 2001, the Internet Engineering Task Force has described a standard for generating proofs of existence for pieces of data at given points in time. The technical specifications of this protocol are contained in IETF RFC 3161 [1].

This works by having a time-stamping service (TSA) give out `TimeStampToken`s for requested hash values of the data. These tokens can then be verified by any third party. If both the signature is valid and the verifier trusts the certificate chain, the verifier can trust that the datum did already exists at the date contained in the token.

TSP need not be in real time. Request-response round trips may take any amount of time. A time-stamp token will only prove that a hash existed before from the moment that the time-stamp existed. So note that TSP has little to do with clock synchronization, and is not comparable to protocols like NTP [4] and Roughtime [3][1].

The main use of being able to prove the existence of some datum at some point in time is to prove that something has not changed since then. A concrete example of this is proving the originality of some intellectual property: When you invent something, you let a TSA generate a token for you. With this token, you may later be able to prove that you did the invention before somebody else did. This may turn out very useful in court.

[1] There *do* exist other methods and protocols analogous to TSP. I will get back to this in section 3.

2 Protocol description

2.1 Building blocks

The TSP protocol is an extension on the X.509 PKI system which is described by RFC 4210 [5][2]. This means that the TSA has a public certificate corresponding to a private key, with which the time-stamps can be verified. Such a certificate is recognized by the `id-kp-timeStamping` identifier which is added to its *Extended Key Usage* field. Just as with any X.509 certificate, the certificate must be trusted by the verifier through a traditional X.509 certificate chain.

The client sends the TSA a hash of the data they want to have time-stamped. The TSA will then verify if the hash that was used is secure enough (this depends on the TSA's security policy). If the request was valid, the TSA then sends back a `TimeStampToken`, which is a signed structure containing the original hash and the time it was time-stamped. After one round-trip, the client has received its proof of existence. If the client sent an invalid request, the TSA will answer with an error message.

It is the TSA's responsibility to generate secure tokens and specify its content security policy (CSP) in a document. The `TimeStampToken` must refer to this policy, which will in practice almost always be an URL. For example in the case of www.freetsa.org, the policies are referred to as follows[3].

```
X509v3 Certificate Policies:
    Policy: itu-t
      CPS: http://www.freetsa.org/freetsa_cps.html
      CPS: http://www.freetsa.org/freetsa_cps.pdf
      (...)
```

Such a policy is composed by the TSA and specifies how the TSA operates. This generally a written document and is not restricted to any enforced guidelines. These policies may therefore vary widely in length and depth[4].

[2] At the moment the TSP protocol was devised the current reference to the X.509 PKI system was RFC 2510. At this moment is has already been obsoleted by RFC 4210.

[3] This snippet has been extracted from the output of an `openssl x509 -text` command.

[4] The FreeTSA CSP for TSP is less than a page long, whereas the Let's Encrypt CSP for TLS web server authentication (http://cps.letsencrypt.org/, accessed Dec 17, 2016) is more than a hundred pages long, specifying concepts like auditing processes and even fire prevention.

In accordance to the TSA's policy, some features are added to allow the TSA to specify some security details. Firstly, the TSA may add an `accuracy` field. If this field is filled in, this tells the verifier that how large the uncertainty of the TSA's time source is. Secondly, there is the `ordering` field. If this field is set to `true`, the TSA states that all time-stamps from this TSA can be ordered based on the time they were generated, regardless of the accuracy of the TSA's clock. Finally, the protocol allows extensions, as a way to add additional kinds of information in the future.

2.2 TSA responsibilities

To be a TSA, you'll have to obey a certain set of rules. The RFC neatly specifies a list of (in the context of this protocol) pretty straightforward rules. Examples of these rules are that the TSA should use a trustworthy source of time and that they should attach a unique integer for each generated time-stamp token.

2.3 Transport methods

The RFC purposely leaves out any standard methods to transport the response and request from and to the TSA. It does some suggestions, like using HTTP, FTP or email. This allows an implementer to use TSP in a flexible manner by customizing the transport method. In practice however, the most commonly implemented transport method is using HTTP, as described by the RFC.

3 Problems and practicality

3.1 Verification

The RFC does not give any instructions specific to TSP about how to verify tokens. The signed structure is of CMS `SignedData` format, which is described by RFC 5652 [2][5].

Just as with any X.509 trust protocol, verification is only allowed if the verifier trusts any (root) certificate in the trust chain. In the case of web server authentication this is practical, because the trusting and

[5] At the time RFC 3161 (the one that describes TSP) was written, the RFC that described the "Cryptographic Message Syntax" was RFC 2630. Since then it has been updated and obsoleted. At the time of writing, RFC 5652 is the most up to date RFC describing CMS.

auditing of CAs is done by browser vendors. In the case of TSP, this is less practical, because at the moment the client generates the time-stamp, it is not known if the verifier will trust the CA in the future.

3.2 Trust expires

Another factor is that a time-stamp is only valid for as long as all private keys in the trust chain are valid. At the moment a certificate gets revoked or expires, *all* previous time-stamps become invalid.

Also, when the validity of a certificate expires, all these time-stamps will have to be re-signed with a new certificate. The resulting proof is of much later date (except if the token was preventively re-signed).

All signatures will eventually expire and at the point of signing you do not yet know if the certificate will be revoked. These properties make the tokens not very reliable. This can be overcome by just collecting backup-tokens from another TSA as to make your existence proofs more redundant. If one TSA fails, there is still a token left which is signed by another TSA.

3.3 Alternatives

Traditionally, if you don't mind spending some money on proving the existence for some pieces of data, there is a very commonly used alternative for TSP: Hashing your data and publishing the hash value in a widely printed journal or newspaper. If the paper is printed and distributed far and wide, this is nigh-on perfectly secure. Forging a such a "time-stamp" would mean recovering all the existing copies of the printed piece and changing them, a task that is not practically doable.

A more modern alternative is making use of the Bitcoin block chain. You generate a Bitcoin transaction and use the opcode `OP_RETURN` to attach the data's hash value to the transaction. In the next ten minutes the transaction will be verified and included in the public ledger. From this point, the hash value will forever exist in the block chain and you will always be able to look up the moment *when* it was included. This makes for a proof that is more reliable than TSP, because it does not depend on a single trusted authority, but a massive decentralized network of peers. Using the Bitcoin method, opposed to using TSP, is also probably a lot cheaper (depending on your TSA). A drawback of this method is that it is a bit slower than TSP. With TSP, time-stamps can be done in real-time. With the Bitcoin method, the delay may be up to ten minutes.

4 Conclusion

TSP is a simple and effective way to generate trusted time-stamps for data. It is however not much used. Moreover the trust of the time-stamps is somewhat limited. But this protocol is currently the only standard protocol for generating trusted time-stamps in real time. If you don't mind waiting on your tokens for a couple of minutes, you may be better off using the Bitcoin block chain.

References

1. Adams, D.C., Pinkas, D.: Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP). RFC 3161 (Aug 2001), https://rfc-editor.org/rfc/rfc3161.txt
2. Housley, R.: Cryptographic Message Syntax (CMS). RFC 5652 (Sep 2009), https://rfc-editor.org/rfc/rfc5652.txt
3. Langley, A.: Roughtime. https://www.imperialviolet.org/2016/09/19/roughtime.html, accessed: 2016-12-16
4. Mills, D.L.: Network Time Protocol (NTP). RFC 958 (Sep 1985), https://rfc-editor.org/rfc/rfc958.txt
5. Mononen, T., Kause, T., Farrell, S., Adams, D.C.: Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP). RFC 4210 (Sep 2005), https://rfc-editor.org/rfc/rfc4210.txt