C.) Specifications of „Academic Signature- Time Stamps“

Overview

Academic Signature time stamps enable the holder of a private/public key pair to certify for a client that the client had been in possession of a document at a certain point of time. The validity of the certification rests on the credibility and trustworthiness of the stamper.

Consequently the stamper is a witness and has a role comparable to a notary public. As such, he/she looses credibility in the stamping procedure at hand, if he/she is in a state of dependence regarding the client or if he/she is a buddy of the client. A neutral professional relationship yields the highest credibility for the stamp.

The client does not necessarily need to disclose the contents of the document to be stamped. The stamper may just stamp a cryptographic hash value of the document to be able to credibly witness to any third party the fact that the client had been in the possession of the document hash value at the time of the stamping. This would be an indirect proof though but would strongly support the claim that the client had knowledge of the underlying document itself.

It is the explicit responsibility of the stamper to manually check the correctness of the time declaration in the stamp. Academic Signature shows and allows to edit the time and date declaration prior to stamping. This is to unambiguously clarify the fact that the validity of the certified date and time solely rests on the credibility of the stamper. The stamper must be ready to testify in court to back up the claim of the client if the stamps validity is disputed by a third party.

Technically the stamp is realized using digital signatures, pseudo random functions and cryptographic hash functions.

Implementation

The C++ code for creating and verifying time stamps is contained in the modules "timestamp.cpp" and "time_verify.cpp" and their header files, respectively.

Generating the timestamp proceeds as follows:

1. A long number hash value "h_doc" of the document to be stamped is calculated.

2. A temporary stamper info file is created. The file contains date and time info, a stampers comment and a reference to the client who requested the stamp. See the contents of an example stamper file below:

```
* ********************************************

Timestamping helpersxx.cpp for_myself
at: 2013-12-16 13:19:59 MEZ
by Prof.Dr.Michael_Anders using_key: anders_256_k1
Comment: just_a_demo
* ********************************************
```

3. A long number hash value "h_time" of the stamper file is calculated. The same algorithm is chosen as for the hash calculation of the document in task 1.

4. Both long number hash values have structurally and accidentally caused leading zeroes removed and are concateneated (h_doc|| h_time). The resulting bytestring "myblock" of length "totsize" is developed using the pseudo random function "develop_o" .
Code excerpt:

```
  .
  .
  .
if( !develop_o( myblock, totsize,2,3,2))
{
  throwout(_("Alarm! Error in timstamp!"),20);
  free(myblock);
  return;
}
  .
  .
  .
```

5. The resulting block is digitally signed (without additional hashing) using Academic Signature's ECDSA method.

6. The stamper info file is prepended with the keyword "stamp_data: " and the result is appended to the signature file. The resulting compound stamp file is renamed to the filename of the document with the additional extension "ects". Ects is the abbreviation of : Elliptic Curve Time Stamp.

A sample stamp file is printed below:

```
Domnam: p256r1
 r: 84b9aab9a8550d11efc98d904a0b599aed3f54f30eccb6832ead7d34e0c5f9d5
 s: 8868fca1dfe6eba56e6fb015525f9da3c9c87e7fabb91403b69362498e826076
Hash: sha4

stamp_data:
* ********************************************

Timestamping helpersxx.cpp for_myself
at: 2013-12-16 13:19:59 MEZ
by Prof.Dr.Michael_Anders using_key: anders_256_k1
Comment: just_a_demo
* ********************************************
```

The verification of the time stamp is done accordingly:

1. The stamper info file is extracted from the stamp file, the split marker being the sequence "stamp_data: ".

2. The hash value of stamper info file and the hash value of the document are calculated.

3. Both hashes are processed the same way as in stamp generation and the validity of the resulting block is checked against the digital signature contained in the leading part of the stamper file.

4. In case the verification was successful, the stamp data is shown to the verifier along with a notice about the validity of the stamp. Otherwise the verifier is informed about the failure of the stamp validation attempt.

As shown above and according to the general guideline, Academic Signature's time stamps are not secret and thus are presented and transferred as human readable ascii files. Their security rests on the security of the elliptic curve crypto system, the security of the used cryptographic hash algorithm and the security of the pseudo random function "develop_o". Their validity rests on the trustworthiness of the stamper.