Notary

A Flask Application for Trusted Timestamping

Dario Gjorgjevski[1] Petar Tonkovikj[1]

[1]Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University in Skopje

October 29, 2017

Contents

Trusted Timestamping

- Trusted timestamping is the process of securely keeping track of the creation time of a document.
- "Securely" roughly means that *no one* — not even the owner of the document — should be able to change it.
- RFC3161 specifies a scheme for trusted timestamping for the PKI
  $\Longrightarrow$ OpenSSL provides an open-source implementation of RFC3161.

Web Application

Architectural choices

- Must ensure independent timestamp generation and verification $\Longrightarrow$ two parts meant to be run in isolated environments.
- Both parts are built using Flask[1] and Python 3.
- Simple and bulletproof architecture built on top of the Web Server Gateway Interface (WSGI).

[1] `http://flask.pocoo.org/`

Web Application

Deployment

- WSGI scripts can be deployed on the Apache HTTP Server via `mod_wsgi`.
- HTTPS and user authentication are provided by OpenSSL via `mod_ssl`.
- The certificate used for SSL is signed by our TA, Hristina.
- The certificate used for trusted timestamping is generated and self-signed by us.
- We allow only users with certificates issued by our organization to authenticate themselves to the website. See listing 1.

Web Application

Authentication

```
#   Access Control:
#   With SSLRequire you can do per-directory access control based
#   on arbitrary complex boolean expressions containing server
#   variable checks and other lookup directives.  The syntax is a
#   mixture between C and Perl.  See the mod_ssl documentation
#   for more details.
<Location>
SSLRequire %{SSL_CLIENT_I_DN_O} in \
           {"Faculty of Computer Science and Engineering", \
            "Ss. Cyril and Methodius University", \
            "FCSE", "UKIM"}
</Location>
```

Listing 1: OpenSSL Access Control

Contents

Generating a Trusted Timestamp

Informally

Generating a trusted timestamp is straightforward and consists of:

1. Hashing the data that needs to be timestamped;
2. Concatenating the hash with the timestamp;
3. Hashing the concatenation of the two;
4. Digitally signing the hash with the private key of the TSA.

Generating a Trusted Timestamp

Formally

A trusted timestamp contains `data`, a `timestamp`, a hash `H`, and a signature `S`. The latter two are defined as:

$$\begin{aligned} \mathtt{H} &:= \mathsf{H}(\mathsf{H}(\mathtt{data}) \| \mathtt{timestamp}), \text{ and} \\ \mathtt{S} &:= \mathsf{Sig}_{\mathtt{TSA}}(\mathtt{H}), \end{aligned} \tag{1}$$

where $\mathsf{Sig}_{\mathsf{pk}}(\cdot)$ is a digital signature under the private key pk, $\mathsf{H}(\cdot)$ is a cryptographic hash function such as SHA-256 or SHA-512, and "$\|$" denotes concatenation.

Verifying a Trusted Timestamp

1. Hash data and tentative timestamp as previously.
2. Verify the TSA's signature on the provided hash.
3. Compare the computed hash and the provided hash.

The two hashes being identical is a strong guarantee of the timestamp being correct (i.e., issued by the TSA) and unaltered.

Contents

Supplying the Authentication Certificate

All major web browsers are able to supply PKI certificates for authentication to the website. An example doing this on our website is found on figure 1.

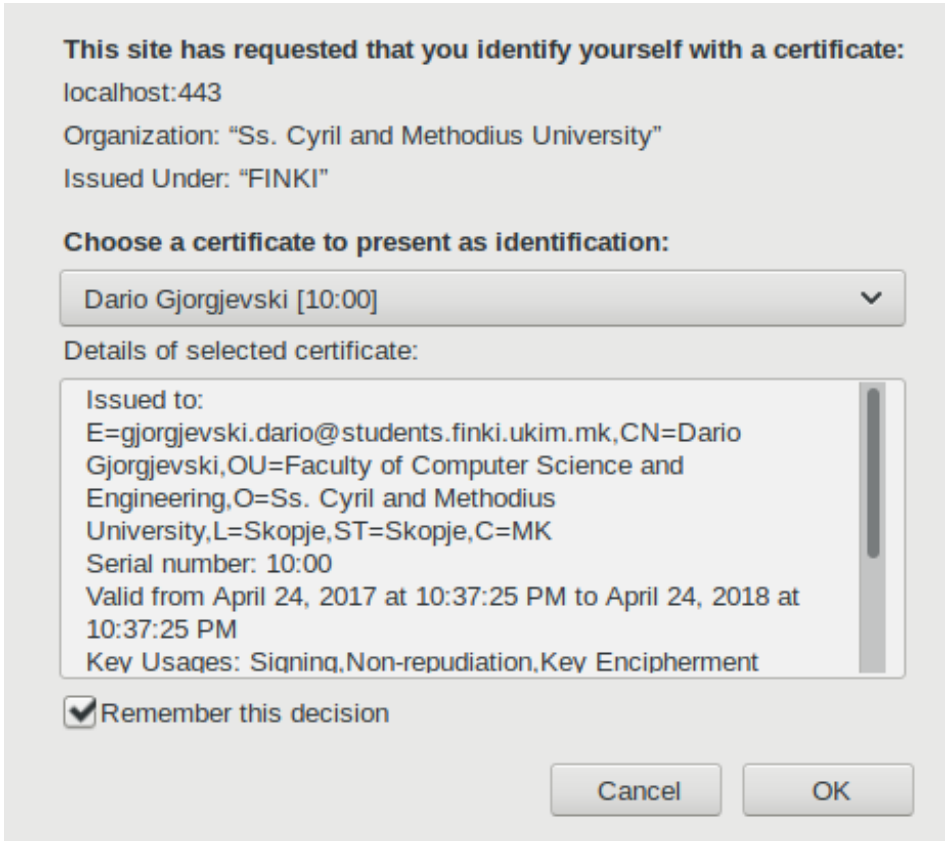

Figure 1: Mozilla Firefox Prompting for Certificate

Verifying the SSL Connection

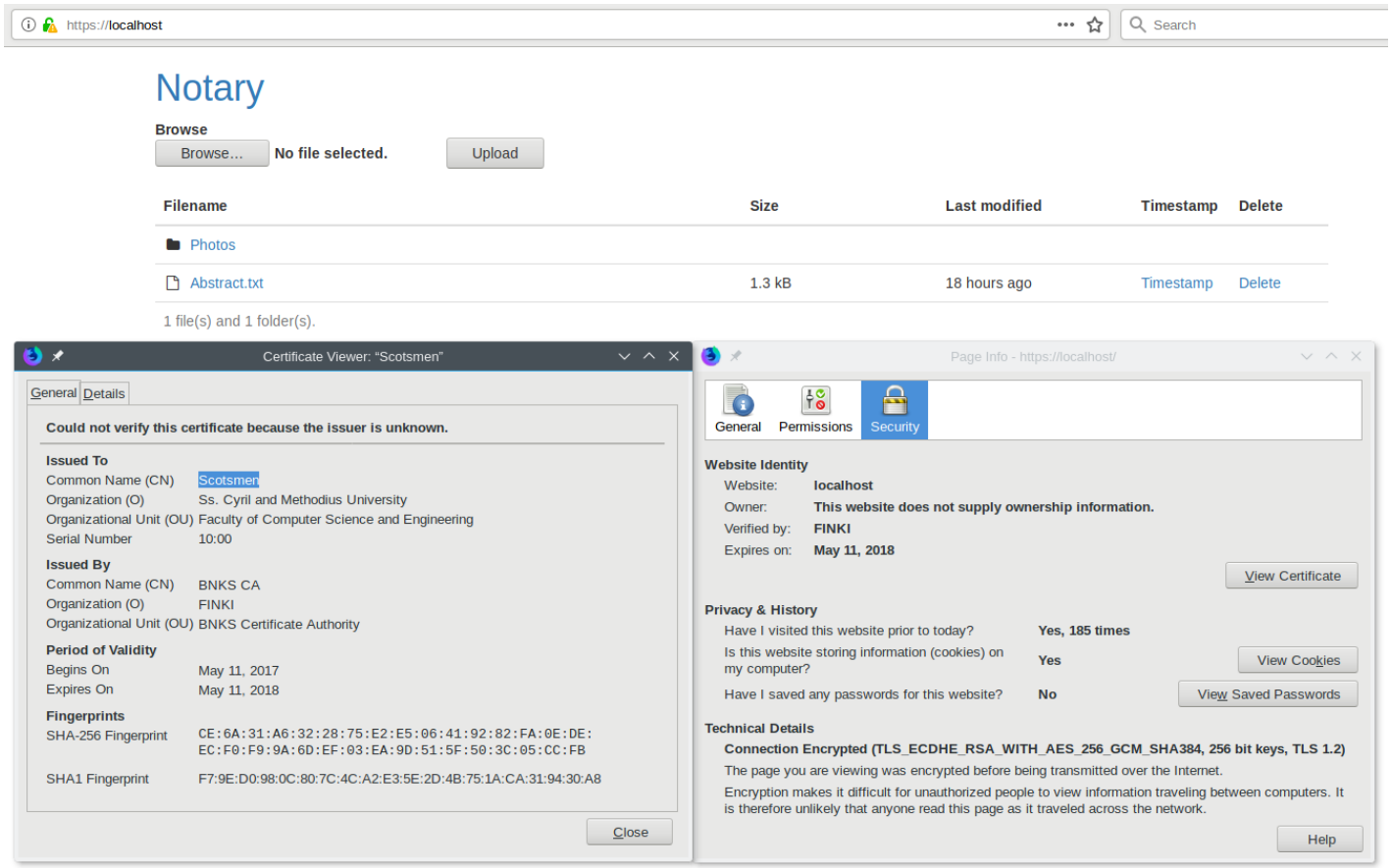

Figure 2: SSL Certificate Information

Uploading Documents for Trusted Timestamping

- Uploading is as simple as pressing the Upload button.
- Once on the server, document can be either:
  - Timestamped; or
  - Deleted.
- Pressing the Timestamp button sends to the user a zip file with a:
  - Timestamp query (TSQ) used to timestamp the file; and a
  - Timestamp response (TSR) containing the trusted timestamp.

Uploading the TSQ and TSR for Verification

The TSQ and TSR files downloaded from the file server can be uploaded to the second web application for verification. Once uploaded, they are passed as input to the `ts -verify` OpenSSL command.

Figure 3: Uploading the Required Files

Inspecting the Verification Results

Once a user has uploaded the TSQ and TSR files, he/she will be redirected to a page containing the results of the verification.

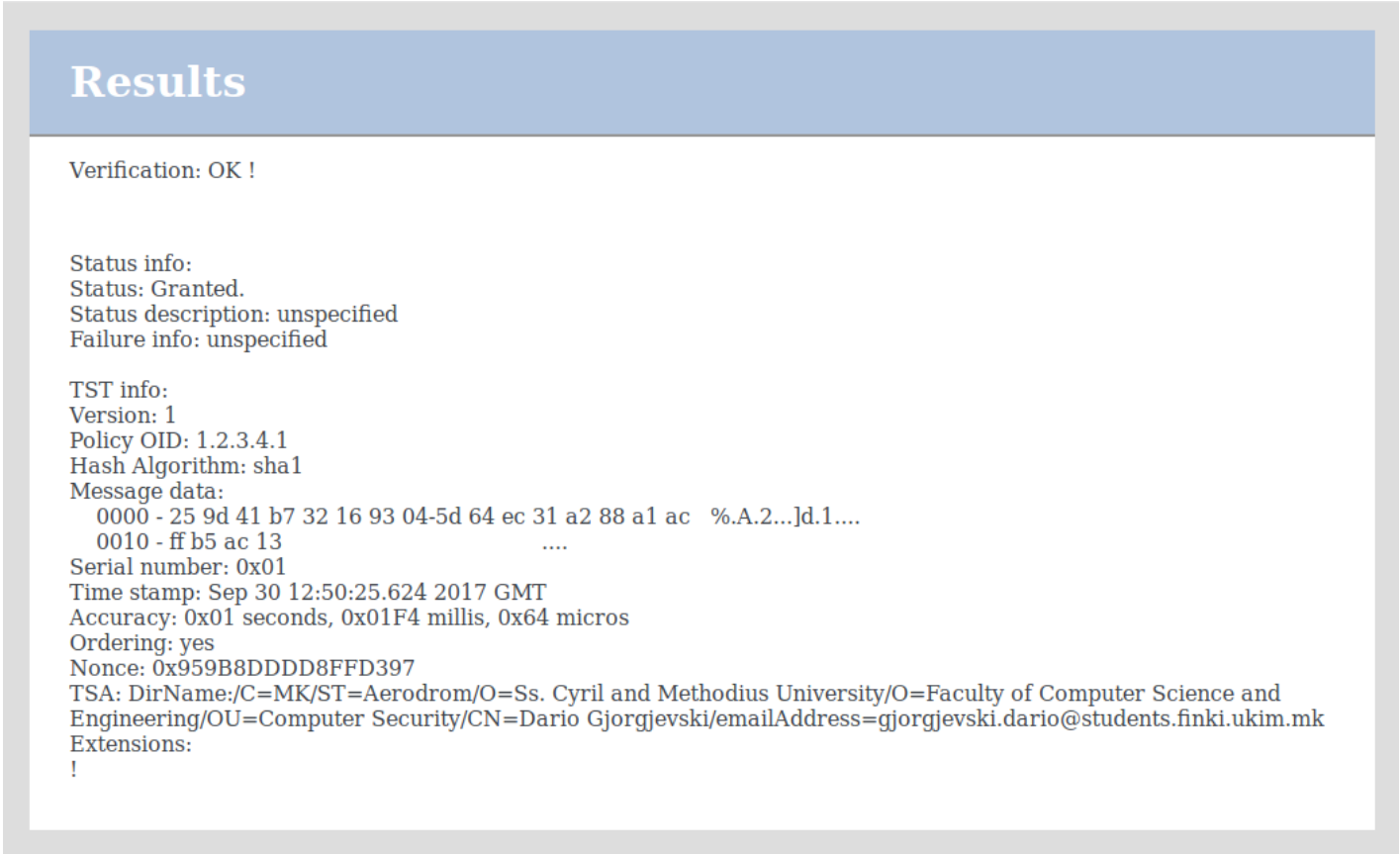

Figure 4: Output of `ts -verify`

Pros and Cons of Having Two Separate Applications

Pros

- Users to not need to use OpenSSL at all $\implies$ more user-friendliness and less error-proneness.
- Separate entities for timestamping and verification $\implies$ more trustworthiness since the compromise of one entity can be "detected" by the other.

Cons

- Deployment: it is harder to deploy and keep two applications up-to-date.
- Convenience: users need to use two different applications for a single functionality.

Contents

Conclusion

- We developed a simple trusted timestamping server.
- Suitable for deployment in *closed* environments: closed in a sense that users can authenticate themselves via certificates.
- All components utilized are open-source, simple, and have been used and studied extensively.

Future Work

- Store files on a database for easier fault-tolerance and backups. At present, files are stored on the file system as-is.
- Functionality to timestamp files the moment they are uploaded rather than upon request.
- Alternative ways of authentications (user accounts).
- Private files.
- Same certificate for the TSA and the SSL connection.