Communication and Security Infrastructure, Spring 2023,

assignments list # 1 – OpenSSL (part 1), 2023-10-03

1. **[3 pts] [Getting OpenSSL]** Download and install OpenSSL [https://www.openssl.org] for your operating system, if it's not installed already. On Linux you should be able to check it with:

```
user@host:~$ openssl version
OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2 15 Mar 2022)
```

Familiarize yourself with available commands and their syntax. In particular note how all options are "long", e.g.

```
user@host:~$ openssl x509 -help
Usage: x509 [options]

General options:
 -help                      Display this summary
 -in infile                 Certificate input, or CSR input file with -req
                            (default stdin)
 -passin val                Private key and cert file pass-phrase source
 -new                       Generate a certificate from scratch
 ...
```

Try using some simple tools and compare their outputs with outputs of other tools or libraries, for instance:

```
user@host:~$ openssl sha256 -hex tmp.txt
SHA256(tmp.txt)= 780e0ded8d9a9d83ab71210d5d0b6c2c5e44ec45dc618613c843588b037e1d73
user@host:~$ sha256sum tmp.txt
780e0ded8d9a9d83ab71210d5d0b6c2c5e44ec45dc618613c843588b037e1d73  tmp.txt
```

2. **[5 pts] [X509 Certificate Verification]** In your internet browser, visit any site supporting HTTPS (for instance https://pwr.edu.pl). Using the browser, download the website SSL/TLS certificate.

   - What form is it in?
   - Do you recognize it?

   Next, use `openssl` to inspect the certificate. In particular try using `openssl x509` command with different options (try `text`). Find out:

   - Who signed the certificate?
   - What is the signature algorithm?

- Who is the certificate for, including which domains? (see **X509v3 Subject Alternative Name** extension)
- What X509 extensions are present?

Finally, use `openssl verify` to check if there is a trusted path against one of the roots of trusts on your computer. **Hint:** you may need to dig a bit for some intermediate certificates.

3. **[12 pts]** [**Certificate Authority**] Familiarize yourself with OpenSSL CA and certificate request facilities (`openssl ca`, `openssl req`). Prepare individual directories for three parties:

   (a) (Root) Certificate Authority A with a self-signed certificate;

   (b) Intermediate CA B with certificate signed by A;

   (c) Service C with certificate signed by B.

   Keep all files "owned" by the parties in their respective directory, simulate sending (for instance a certificate request) by copying the file from one directory to another. Make sure the private keys never leave their owners(' directories)!

   Write/generate CA configurations for parties A and B. Generate certificate signing requests or self-signed certificates, where applicable. Verify and then sign the requests. In the end, you should have three separate PEM-encoded certificates:

   - Self-signed certificate for A;
   - A-signed certificate for B;
   - B-signed certificate for C.

   For the service C use Subject Alternative Name (SAN) corresponding to a DNS domain you have control over, at least `localhost` (you can use any domain you like).

   Combine all certificates into a single certificate chain ("owned" by C).

   Using your software of choice try hosting a small website on that domain, providing the combined chain and C's private key as SSL/TLS context. For example, Flask can serve HTTPS by:

   ```
   app.run(ssl_context=('path/to/cert', 'path/to/key'))
   ```

   Try connecting to the website with your browser (you may need to force it to use HTTPS with localhost). What kind of error are you getting? Try adding the self-signed A certificate to your certificate store. Does it work now?

   What are the results?

/-/ Marcin Słowik