Data Integrity

NDBI046: Practical class 12

Use case

- We have prepared a data catalogue including distributions of our datasets
- We want to *publish the data catalogue* while *ensuring* their *integrity*

Exercise 12.1: Assurance of integrity and authenticity of distributions

- Update the catalog description from Exercise 10.6 in order to *secure distribution integrity*
  - In particular, extend the catalog description by spdx:checksum property and spdx:Checksum class to provide digest for DCAT distributions
    - spdx:checksum provides a mechanism to verify that the contents of a file have not been changed
    - spdx:Checksum is a value that allows to check the integrity of the contents of a file
- Save data catalog description into file data-catalog.ttl
- For more information, see the DCAT vocabulary documentation: https://www.w3.org/TR/vocab-dcat-3/#security_and_privacy

Tip: To hash the contents of the file, use the sha1 algorithm from the hashlib module

OpenSSL and certificates

OpenSSL

- An open source *software library* that provides *cryptographic functions and protocols*, including SSL/TLS
- Widely used for secure network communications and for implementing cryptographic protocols in various applications

Certificates

- Digital *documents* used to *establish the identity* of individuals, servers or organisations in online communications
  - Contain information such as the identity of the subject, the public key and the digital signature, issued by a trusted certificate authority (CA)
- Play a crucial role in ensuring secure communications by verifying the authenticity and integrity of the parties involved

Certificate signing request (CSR)

- A *request* generated by an entity (such as a server or user) *to obtain a digital certificate from a certification authority*
  - Contains information such as the public key and the identification details of the subject
- The CSR is *signed with the private key* of the subject, demonstrating control over the public key

Private key

- A cryptographic key that is kept secret and used to generate digital signatures and decrypt data
- It is *paired with the public key* to create a key pair for asymmetric encryption

Example 12.2: Create a certificate signing request

- Using openssl, *create* a new *RSA private key*
  - Use *256-bit AES* encryption
  - Specify the name of the output file to which the newly generated private key will be saved
  - Specify the *key length* in bits
    - Usually *2048 bits* are used in practice
  - When prompted, enter the *passphrase*

- Then use openssl to *create a certificate signing request* from our existing private key
  - Provide the following CSR information:
    - *countryName* (C), *organizationName* (O), and *commonName* (CN)
  - Certificate request has the file extension *.csr

Example 12.2: Create a certificate signing request (Solution)

private key is encrypted using AES with 256 bits

generate a new RSA key

```
openssl genrsa \
  -aes256 \
  -out private.key \
  2048
```

the name of the output file

the key length in bits

the path to an existing private key that will be used to generate the CSR

perform a CSR-related operation

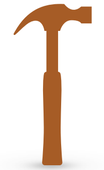
```
openssl req \
  -key private.key \
  -new \
  -out request.csr \
  -subj "/C=CZ/O='Charles University'/CN=ksi.mff.cuni.cz"
```

create new CSR

name of the output

subject information for the certificate

Example 12.2: Create a certificate signing request (Solution)



- *Certificate authority will sign the CSR* with the root CA certificate and its private key
  - The user usually does not have access to this, hence *the step must be requested from the CA*

```
3 openssl x509 \
    -req \
    -CA ca_root_certificate.crt \
    -CAkey ca_private.key \
    -in request.csr \
    -out certificate.crt \
    -days 365 \
    -CAcreateserial
```

- Alternatively, you may create a *self-signed certificate*

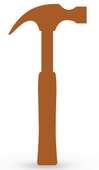
```
4 openssl req -x509 \
    -newkey rsa:4096 \
    -sha256 \
    -nodes \
    -keyout private.key \
    -out certificate.crt \
    -subj "/C=CZ/O='Charles University'/CN=ksi.mff.cuni.cz"
```

Example 12.3: Sign the data catalog file

- Use openssl to *sign data catalog file* with your certificate
  - Use the *sha256* algorithm *for hashing*
  - *Encode* binary content *using base64* format

- Save the digital signature in the file data-catalog.sha256.sign

Example 12.3: Sign the data catalog file (Solution)



hash calculation tool (digest)

hashing algorithm sha256

the path to the private key

name of the output file where the output hash is stored

name of the input file for which the hash is calculated

```
5 openssl dgst \
    -sha256 \
    -sign private.key \
    -out data-catalog.sha256 \
    data-catalog.ttl
```

binary data conversion to base64 format

input file name

output file name

```
6 openssl base64 \
    -in data-catalog.sha256 \
    -out data-catalog.sha256.sign
```

Exercise 12.4: Publish data

- *Publish* your *data fragments* on the server webik.ms.mff.cuni.cz
  - Add an index.html page with links to the following files:
    - Certificate certificate.crt
    - Data catalog description data-catalog.ttl
    - Digital signature data-catalog.sha256.sign
    - CSV dataset distribution waste_dataset.csv
    - Data cube distribution waste_cube.ttl

- Do not publish your private key!

References

DCAT Vocabulary

- Security and privacy: https://www.w3.org/TR/vocab-dcat-3/#security_and_privacy
- IANA media types: https://www.iana.org/assignments/media-types/media-types.xhtml

OpenSSL

- OpenSSL: https://www.openssl.org/
- Manual: https://www.openssl.org/docs/manmaster/man1/openssl.html

Python

- hashlib: https://docs.python.org/3/library/hashlib.html

HTML

- W3C HTML tutorial: https://www.w3schools.com/html/