How (not) to make a mockery of trust

Testing client software for public-key infrastructure (FOSDEM ’22)

Matthias Valvekens

2022–02–05

About me

- I’m a research engineer at iText (PDF industry)
- FOSS development is my job and my hobby
- I like digital signatures
- GitHub: `MatthiasValvekens`
- Website: `mvalvekens.be`

A testing problem

Recap: digital signatures

- Uses public-key cryptography: a *pair* of keys
  - Private key: used for signing
  - Public key: used for validation
  - Security backed by fancy maths
- Many workflows require *certificates*
  - Binds the signer's public key to their identity.
  - Issued by a *certificate authority*
  - Common standard: X.509 (and profiles thereof)
  - Typically attached to or embedded into the signed payload

Maths: the easy part of validation?

Cryptographic maths:

- Tells you whether a signature was produced with a particular key.
- Just use a good cryptographic library.

The hard part is *not* mathematical integrity, but this:

Question

Who is actually in control of the signing key? Is the signer who they claim to be?

The vast majority of real-world PKI engineering problems actually deal with *this* question.

The role of the certificate authority (CA)

- Issues certificates stating the "owner" of a key. Examples:
  - ID card: subject is a natural person.
  - TLS certificate: subject is a domain name.
- NB: in practice, there are often multiple CA "layers" ($\rightsquigarrow$ *chain of trust*).

The role of the certificate authority (CA)

Workflow:

- The subject must prove to the CA that they control the key
  - ⇝ requires a degree of **trust**!
- Certificates are limited in time for this reason.

Question

But what if the key is compromised during the lifetime of the certificate?

Revocation

- CAs can *revoke* certificates they issued
- Can happen for many reasons:
  - Key compromise, suspicion of fraud, cessation of operation, …
- Part of certificate validation involves checking revocation status
- Particularly important for signatures, also relevant for TLS

Mechanisms to convey revocation information:

- CRL: **C**ertificate **R**evocation **L**ist
- OCSP: **O**nline **C**ertificate **S**tatus **P**rotocol

These services are provided by the CA, usually over the internet.

Note: signers also have to deal with this in some cases (LTV signatures)

Software engineering implications

If you're designing an application that needs to

- validate digital signatures, or
- produce digital signatures with long lifetimes

…then you'll need to interact with those trust services!

Question

How do you even begin to test such a thing?

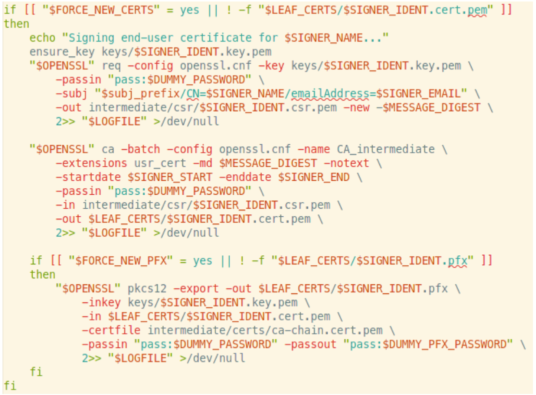
Testing strategies

It’s about more than just test data

- Generation of certificates and CRLs can be scripted
  - some `bash`-fu with `openssl` + `faketime` gets you quite far
  - Result can be hard to maintain
- What about mocking online services?
  - OCSP responders (RFC 6960)
  - Time-stamping services (RFC 3161)
- Testing failure cases: what about…
  - …generating certificates that are broken in particular ways?
  - …revocation scenarios?
  - …validating certificate extensions that `openssl` doesn’t handle?

Illustration: an excerpt from my previous testing setup

Just a matter of knowing the right `openssl` incantations, eh?



```
if [[ "$FORCE_NEW_CERTS" = yes || ! -f "$LEAF_CERTS/$SIGNER_IDENT.cert.pem" ]]
then
    echo "Signing end-user certificate for $SIGNER_NAME..."
    ensure_key keys/$SIGNER_IDENT.key.pem
    "$OPENSSL" req -config openssl.cnf -key keys/$SIGNER_IDENT.key.pem \
        -passin "pass:$DUMMY_PASSWORD" \
        -subj "$subj_prefix/CN=$SIGNER_NAME/emailAddress=$SIGNER_EMAIL" \
        -out intermediate/csr/$SIGNER_IDENT.csr.pem -new -$MESSAGE_DIGEST \
        2>> "$LOGFILE" >/dev/null

    "$OPENSSL" ca -batch -config openssl.cnf -name CA_intermediate \
        -extensions usr_cert -md $MESSAGE_DIGEST -notext \
        -startdate $SIGNER_START -enddate $SIGNER_END \
        -passin "pass:$DUMMY_PASSWORD" \
        -in intermediate/csr/$SIGNER_IDENT.csr.pem \
        -out $LEAF_CERTS/$SIGNER_IDENT.cert.pem \
        2>> "$LOGFILE" >/dev/null

    if [[ "$FORCE_NEW_PFX" = yes || ! -f "$LEAF_CERTS/$SIGNER_IDENT.pfx" ]]
    then
        "$OPENSSL" pkcs12 -export -out $LEAF_CERTS/$SIGNER_IDENT.pfx \
            -inkey keys/$SIGNER_IDENT.key.pem \
            -in $LEAF_CERTS/$SIGNER_IDENT.cert.pem \
            -certfile intermediate/certs/ca-chain.cert.pem \
            -passin "pass:$DUMMY_PASSWORD" -passout "pass:$DUMMY_PFX_PASSWORD" \
            2>> "$LOGFILE" >/dev/null
    fi
fi
```

Enter Certomancer!

At some point, I got sufficiently annoyed to build a solution: **Certomancer**

- Generates test certificates and CRLs
- Spins up live on-line test services
- Declarative YAML config (no scripting required)
- Python extension API

Licence: MIT

GitHub: `MatthiasValvekens/certomancer`

Using Certomancer

Take a look around

1 Pre-generate some key pairs

```
$ openssl ecparam -name secp384r1 -genkey -out ecdsa-test.key.pem
$ openssl genrsa -out rsa-test.key.pem
Generating RSA private key, 2048 bit long modulus (2 primes)
.................................................................+++++
......................................................+++++
e is 65537 (0x010001)
$ ls -alh
total 8.0K
drwxr-xr-x  2 matthias matthias   80 Jan 21 13:10 .
drwxrwxrwt 25 root     root      780 Jan 21 13:05 ..
-rw-------  1 matthias matthias  359 Jan 21 13:10 ecdsa-test.key.pem
-rw-------  1 matthias matthias 1.7K Jan 21 13:10 rsa-test.key.pem
$
```

Take a look around

1. Pre-generate some key pairs
2. Write YAML config:
   - Define entities

```yaml
external-url-prefix: "http://ca.example.com"
keysets:
  testing-ca:
    path-prefix: /tmp/keys
    root: { path: root.key.pem }
    interm: { path: interm.key.pem }
pki-architectures:
  testing-ca:
    keyset: testing-ca
    entity-defaults:
        country-name: BE
        organization-name: TestCA
    entities:
      root: { common-name: Root CA }
      interm: { common-name: Interm CA }
    certs: ...
    services: ...
```

Take a look around

1. Pre-generate some key pairs
2. Write YAML config:
   - Define entities
   - Define certs

```yaml
# Sample certificate definition
interm:
    issuer: root
    validity:
      valid-from: "2000-01-01T00:00:00+0000"
      valid-to: "2100-01-01T00:00:00+0000"
    extensions:
      - id: basic_constraints
        critical: true
        value:
          ca: true
          path-len-constraint: 0
      - id: key_usage
        critical: true
        smart-value:
          schema: key-usage
          params: [digital_signature, key_cert_sign]
      - id: crl_distribution_points
        smart-value:
          schema: crl-dist-url
          params: {crl-repo-names: [root]}
```

Take a look around

1. Pre-generate some key pairs
2. Write YAML config:
   - Define entities
   - Define certs
   - Define services

```yaml
services:
  ocsp:
    interm:
      for-issuer: interm
      responder-cert: interm-ocsp
      signing-key: interm-ocsp
  crl-repo:
    root:
      for-issuer: root
      signing-key: root
      simulated-update-schedule: "P30D"
  time-stamping:
    tsa:
      signing-key: tsa
      signing-cert: tsa
```

Take a look around

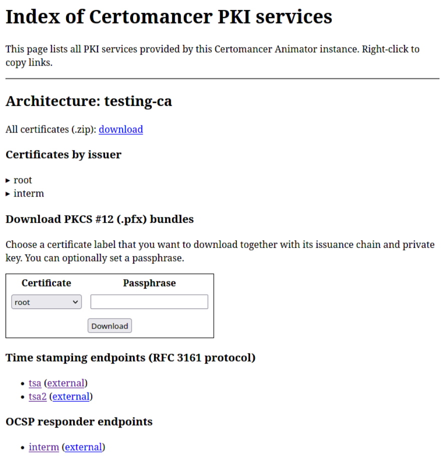
1. Pre-generate some key pairs
2. Write YAML config:
   - Define entities
   - Define certs
   - Define services
3. Run `certomancer animate`

```
$ certomancer --config example.yml animate
 * Running on http://127.0.0.1:9000/ (Press CTRL+C to quit)
127.0.0.1 - - [21/Jan/2022 13:52:52] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [21/Jan/2022 13:52:53] "GET /favicon.ico HTTP/1.1" 404 -
$
```

Take a look around

1. Pre-generate some key pairs
2. Write YAML config:
   - Define entities
   - Define certs
   - Define services
3. Run `certomancer animate`
4. Go look at `http://localhost:9000`

CLI demo on Asciinema: `asciinema.org/a/406798`



Index of Certomancer PKI services

This page lists all PKI services provided by this Certomancer Animator instance. Right-click to copy links.

Architecture: testing-ca

All certificates (.zip): download

Certificates by issuer

▸ root
▸ interm

Download PKCS #12 (.pfx) bundles

Choose a certificate label that you want to download together with its issuance chain and private key. You can optionally set a passphrase.

Certificate	Passphrase
root	

Download

Time stamping endpoints (RFC 3161 protocol)

- tsa (external)
- tsa2 (external)

OCSP responder endpoints

- interm (external)

How I use Certomancer

In personal projects:

- with `requests-mock` in `pytest` tests;
- with `aiohttp` to create mocks for the Cloud Signature Consortium API.

At work:

- as part of a network service that spawns PKI architectures with on-demand config for integration tests;
- when developing proof-of-concept implementations for standards work;
- for product demonstrations, troubleshooting and internal training purposes.

Thanks for listening!

Questions?

Certomancer

github.com/MatthiasValvekens/certomancer

iText

kb.itextpdf.com/home