- **User kommt mit Benutzerdaten**
  - ~~1.~~ privaten Schlüssel generieren
    - `OpenSSL.crypto.PKey().generate_key(type, bits)`
      - https://skippylovesmalorie.wordpress.com/2010/02/12/how-to-generate-a-self-signed-certificate-using-pyopenssl/
  - ~~2.~~ öffentlichen Schlüssel zu dem privaten Schlüssel generieren
  - ~~3.~~ Benutzerdaten und öffentlichen Schlüssel in ein X509-Objekt verpacken
  - ~~4.~~ Zertifikat zu dem X509-Objekt erzeugen
  - ~~5. X509-Objekt und privaten Schlüssel an den Benutzer zurück schicken~~
- **Server kommt mit Daten und selbst generiertem Schlüssel**
  1. öffentlichen Schlüssel und Serverdaten in ein X509-Objekt verpacken
  2. Zertifikat zu dem X509-Objekt erzeugen
  3. X509-Objekt an den Server zurück schicken

API:
→ https://pyopenssl.readthedocs.org/en/latest/api/crypto.html#x509-objects

UC1: Zertifikat für Benutzer erstellen

Benutzer schickt uns Benutzerdaten:

- Land (C)
- Bundesland/Staat (ST)
- Stadt (L)
- Firma (O)
- Organisation (OU)
- …

Die Daten werden später zur Generierung eines Zertifikates gebraucht.

Folgende Objekte werden in diesem Prozess erzeugt:

- PKey → Schlüsselpaar
- X509 → Zertifikat: hält PKey.pubKey, Benutzerdaten, evtl. Extensions, Gültigkeiten, …
- PKCS12 → X509-Object, Pkey.privKey

Ablauf

1. Generierung des Schlüsselpaares (PKey-Object):
```
k = crypto.PKey()
k.generate_key(crypto.TYPE_RSA, 2048)
```
2. `Zertifikat erzeugen (X509-Object) → Benutzerdaten, Adresse der CA, Gültigkeiten, pub-Key des Benutzers und mit privatem Schlüssel signieren:`

```
cert = crypto.X509()
cert.get_subject().[C/ST/...] = „erhaltene Benutzerdaten“ # X509Name-Obj
cert.get_subject().CN = gethostname() # Adresse der CA
cert.gmtime_adj_notBefore(0)
cert.gmtime_adj_notAfter(10*365*24*60*60)
cert.set_issuer(cert.get_subject()) # ACHTUNG: self-signed!!!
cert.set_pubkey(k)
cert.sign(k, 'sha1')
```

3. PKCS12-Object erstellen → Kombination aus dem Zertifikat (X509) und dem privaten Schlüssel des Benutzers:
```
pkcs = crypto.PKCS12()
pkcs.set_certificate(cert) # Wenn nur root-CA!
pkcs.set_privatekey(k)
# set_friendlyname ist wofür???
```

UC2: Zertifikat signieren

Server schickt uns Benutzerdaten und public Key in Form einer Signing Request

Folgendes Objekt erwarten wir von der RA:

X509Req → enthält Benutzerdaten und public Key des Servers

Folgendes Objekt schicken wir an die RA zurück:

PKCS12 → enthält zu gesendete Daten inklusive der CA Daten und dem Zertifikat

Ablauf

1. Daten aus X509Req in X509-Object kopieren
```
cert = crypto.X509()
cert.set_subject(req.get_subject())
cert.set_pubkey(req.get_pubkey())
```
2. Eigene Issuer Daten in das X509-Object schreiben
```
cert.get_issuer().C = „DE“
…
```
3. Gültigkeit in dem X509-Object setzen
```
cert.set_notBefore(...)
cert.set_notAfter(...)
```
4. X509-Object mit unserem private Key signieren
```
cert.sign(privKey, „sha1“)
```
5. PKCS12-Object zum Versenden mit erstellten Daten füllen
```
pkcs = crypto.PKCS12()
pkcs.set_certificate(cert)
```
6. Daten an die RA zurück senden