openssl für alle Tage

Schlüssel erzeugen

```
openssl genrsa -aes256 -out dvsdnet.devoteam.de.key.pem
```

Bei RSA Schlüsseln wird automatisch ein Passwort abgefragt. Das kann man in einem zweiten Schritt wieder entfernen:

```
openssl rsa -in ${filename}.key -out ${filename}.key
```

ECDSA (elliptic curve)

Diese Schlüssel sind kleiner, bein Erzeugen wird kein Passwort verlangt

```
openssl ecparam -name prime256v1 -genkey -noout -out key.pem
```

ec Key **mit** Passwort erzeugen

```
openssl ecparam -name prime256v1 -genkey | openssl ec -aes256 -out key.pem
```

Certificate Signing Request

Standard

Der csr wird entweder selber signiert oder an eine externe CA gegeben. -days regelt die Gültigkeit des Zertifikats und überschreibt die default Werte der ssh Konfiguration.

Einfacher CSR

```
openssl req -nodes -new -newkey rsa:2048 -sha256 -out csr.pem
```

```
''openssl req -new -key private/openvpn_client_odysseus.key -out
certs/openvpn_client_odysseus.req.pem -days 730 ''
```

Achtung bei openvpn: CN muß mit "locutusvpn" beginnen, da die im Server geprüft wird (Server Parameter –verify-x509-name locutusvpn name-prefix)

Mit separater cnf Datei

```
[ req ]
default_bits           = 2048
default_keyfile        = privkey.key
distinguished_name     = req_distinguished_name
attributes             = req_attributes
```

```
req_extensions         = v3_ca

dirstring_type = nobmp

[ req_distinguished_name ]
C                      = DE
ST                     = Baden-Wuerttemberg
L                      = Stuttgart
O                      = Mercedes-Benz Group AG
CN                     = Common Name
emailAddress           = test@email.address

[ req_attributes ]
challengePassword              = A challenge password
challengePassword_min          = 4
challengePassword_max          = 20

[ v3_ca ]

subjectKeyIdentifier = hash
basicConstraints = CA:false
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
```

Signieren über eigene CA

`openssl x509 -req -in certs/openvpn_client_odysseus.req.pem -CA certs/RootCA.cert.pem -CAkey private/RootCA.key.pem -out certs/openvpn_client_odysseus.cert.pem -days 720 Achtung: Beim Signieren über die eigene CA werden die Zertifikat-Nr hochgezählt (serial in srl Datei). Die srl Datei muss auch für die ServerCA separat existieren.` mv newcerts/01.pem certs/ cd certs ln -s 01.pem `openssl x509 -hash -noout -in 01.pem`.0

Checks

Inhalt kontrollieren

Auf der Shell im Klartext lesen: bei CERTs mit "x509", bei CSRs mit "req"

```
openssl x509 -text -noout -in ca.crt
openssl req -text -noout -verify -in csr.pem
```

Check: Passen key und crt zusammen?

Replace <public.crt> with the filename of the public certificate.

```
openssl x509 -noout -modulus -in <public.crt> | openssl md5> /tmp/crt.pub
openssl rsa -noout -modulus -in <private.key> | openssl md5> /tmp/key.pub
```

Danach ein diff der beiden Dateien:

```
diff /tmp/crt.pub /tmp/key.pub
```

If nothing is printed to the console, they were found to be a pair. Any differences are printed to the console in detail.

Formate und Konvertierung

x509 nach pem

```
openssl x509 -in certificatename.cer -outform PEM -out certificatename.pem
```

Von pfx nach pem

```
openssl pkcs12 -in [pkcs-12-certificate-and-key-file] -out [pem-certificate-and-key-file]
```

PKCS Dateien erzeugen (*.pfx, *.p12)

```
openssl pkcs12 -export -out certificate.pfx -inkey privateKey.key -in certificate.crt -certfile more.crt
```

ca-certs extrahieren

```
openssl pkcs12 -in ${filename}.pfx -nodes -nokeys -cacerts -out ${filename}-ca.crt
```

key extrahieren

```
openssl pkcs12 -in ${filename}.pfx -nocerts -out ${filename}.key
```

cert extrahieren

```
openssl pkcs12 -in ${filename}.pfx -clcerts -nokeys -out ${filename}.crt
```

ca-certs und cert zusammen kleben

```
cat ${filename}.crt ${filename}-ca.crt> ${filename}-full.crt
```

Passphäse aus key löschen

```
openssl rsa -in ${filename}.key -out ${filename}.key
```

CA und Sub CA erstellen

Anleitung aus http://fra.nksteidl.de/Erinnerungen/OpenSSL.php

Privaten Schlüssel für CA mit gutem PW erzeugen

```
openssl genrsa -aes256 -out /etc/sslprivate/RootCA.key.pem -rand
/etc/ssl/private/.rand 2048
```

Danach öffentlichen Schlüssel erzeugen:

```
openssl req -new -x509 -days 1827 -key /etc/ssl/private/RootCA.key.pem -out
/etc/sslcert /RootCA.cert.pem
```

Mit der ServerCA lassen sich jetzt Zertifikate für Server erstellen. Diese sollten - weil sie ihren Private Key ohne menschliches Zutun benutzen müssen - keine Passphrase haben. Dies geschieht durch Weglassen des Verschlüsselungs-Algoritmus beim Erstellen des Private Key. Ansonsten sind das auch nur normale Zertifikate. Allerdings werden sie nicht von der RootCA, sondern von der ServerCA signiert.

CA anlegen

openssl genrsa -out apache.key.pem -rand ./private/.rand 2048 openssl req -new -key apache.key.pem -out apache.req.pem

openssl ca -name ServerCA -in apache.req.pem -out apache.cert.pem

mv newcerts/01.pem certs/

cd certs ln -s 01.pem

openssl x509 -hash -noout -in 01.pem`

Dabei nacheinander Land (DE), Region (ST), Stadt, Organisationsname (O) und -einheit (OU) eingeben, Wichtig: Der "common name" (CN) ist der Apache Server-Name, auf den die Clients zugreifen. Dann die erzeugten *.key, *.crt und *.pem Dateien in die passenden ssl/ Verzeichnisse unterhalb /etc/httpd/ oder /etc/apache2/ verschieben. Bei virtual hosts prüfen, ob ein Eintrag "VirtualHost _default_:443" mit den passenden SSL Regeln besteht.

Achtung: Eigene Zertifikate für die virtual hosts gehen nur mit eigener IP Adresse für jeden dieser Server. Alternative ist subjectaltname, im Zertifikat, ist aber nicht richtig sicher. Dann den Server neu starten. Beim ersten https Zugriff per Browser wird das Zertifikat angeboten. Ggf. den Inhalt des Zertifikats prüfen. (Klick auf das Verschlüsselungs-Symbol im Browser unten rechts)

Verlinkung der erzeugten Zertifikate

Um dieses erste Zertifikat der RootCA in das normale Handling der CA zu geben, muß es noch kopiert und verlinkt werden. Die Zertifikate werden im Unterverzeichnis certs mit dem Namen Ihrer

Seriennummer gelegt und der Hash-Wert wird verlinkt:

cd /etc/sslcerts/
cp RootCA.cert.pem /usr/lib/ssl/RootCA/certs/00.pem

cd /etc/ssl/certs/
ln -s 00.pem `openssl x509 -hash -noout -in 00.pem`.

Sub CA für Server Signaturen erstellen

Wie oben, nur wird ein CSR erzeugt, der von der Root CA signiert werden muss.

privaten Schlüssel erzeugen

Dann singning request (CSR) * mit RootCA signieren

CSR erzeugen: openssl req -new -key /usr/lib/ssl/ServerCA/private/ServerCA.key.pem -out /usr/lib/ssl/ServerCA/ServerCA.req.pem

Signieren und hier die zeitliche Gültigkeit definieren:

openssl x509 -req -days 720 -in certs/ServerCAnetzwissen.req.pem -CA certs/RootCAnetzwissen.cert.pem -CAkey private/RootCAnetzwissen.key.pem -out certs/ServerCAnetzwissen.cert.pem

Achtung: im Verz. des zum Signieren benutzten RootCA files muss es eine serial Datei geben:

The default filename consists of the CA certificate file base name with .srl appended. For example if the CA certificate file is called mycacert.pem, it expects to find a serial number file called mycacert.srl.

mTLS mit Apache

https://vignesh-thirunavukkarasu.medium.com/mtls-with-apache-http-server-fbfd702106ca

EASYrsa (openvpn)

Siehe easyrsa

SSL Test mit openssl s_client

Das Standard-Tool für die Analyse von SSL-Funktionen ist das Kommandozeilenprogramm von OpenSSL. Siehe https://docs.openssl.org/1.0.2/man1/s_client/

```
openssl s_client -connect imap.1und1.de:993
```

Kommt als Ergebnis eine Ciphersuite heraus, die mit DH oder ECDH beginnt, haben sich die beiden Kommunikationspartner auf **Forward Secrecy** geeinigt. Ob ein Server überhaupt Forward Secrecy beherrscht, verrät:

```
openssl s_client -cipher 'ECDH:DH' -connect login.live.com:443
```

Ob ein Server Diffie-Hellman erzwingt, auch wenn der Client RSA bevorzugt, verrät die cipher-Spezifikation 'RSA:ECDH:DH'. OpenSSL unterstützt auch Verbindungen, bei denen man die Verschlüsselung über den starttls anfordern muss, wie es etwa im Mail-Versandprotokoll SMTP üblich ist:

```
openssl s_client -starttls smtp -connect smtp.gmx.net:587
```