Praktikum Nr.3 PKI mit OpenSSL

In diesem Praktikum soll die Open Source Software *OpenSSL* unter Windows eingesetzt werden. Es sollen die unterschiedlichen Einsatzmöglichkeiten vorgestellt werden. Zuerst wird gezeigt, wie die Software eingesetzt werden kann, dann werden Sie selbstständig mit ihr arbeiten.
Sie erstellen eine Zertifizierungsstelle (*infoCA*), die Zertifikate erstellen und zertifizieren kann.

Was ist OpenSSL?

OpenSSL ist ein kryptographisches Toolkit, das die Protokolle Secure Socket Layer (SSL v2/v3), Transport Layer Security (TLS v1) und benötigte und zugehörige kryptographische Standards beinhaltet.

Das Programm *Openssl* ist ein Kommandozeilentool, das die kryptographischen Funktionen der OpenSSL Kryptobibliothek in einer Shell verwendet. Es kann zu folgenden Zwecken verwendet werden:

- Erzeugen von RSA, DH und DSA Schlüsseln
- Erzeugen von X.509 Zertifikaten, CSRs und CRLs
- Betreiben einer CA
- Berechnen von Message Digests
- Ver- und Entschlüsseln von Texten
- SSL/TLS Client und Server Tests
- Bearbeiten von S/MIME signierten oder verschlüsselten E-Mails

1. Installation von OpenSSL

Starten Sie den VMWare Server.
Starten Sie **VM-Client1**.
Installieren Sie OpenSSL durch den Start des Programms **Win32OpenSSL-0_9_8g.exe** aus dem Verzeichnis *c:\IS-Praktikum\OpenSSL*.

Hinweis: Falls es dort nicht vorhanden ist, laden Sie es von
http://www.slproweb.com/products/Win32OpenSSL.html herunter.

Verwenden Sie als Zielverzeichnis der Installation *c:\OpenSSL* an (Standardwert).

Öffnen Sie eine DOS-Box und wechseln Sie ins Installationsverzeichnis c:\OpenSSL. Geben Sie folgenden Befehl ein:

```
C:\OpenSSL>openssl help
openssl:Error: 'help' is an invalid command.

Standard commands
asn1parse  ca        ciphers   crl       crl2pkcs7
dgst       dh        dhparam   dsa       dsaparam
ec         ecparam   enc       engine    errstr
gendh      gendsa    genrsa    nseq      ocsp
passwd     pkcs12    pkcs7     pkcs8     prime
rand       req       rsa       rsautl    s_client
s_server   s_time     sess_id   smime     speed
spkac      verify    version   x509

Message Digest commands (see the `dgst' command for more details)
md2        md4       md5       rmd160    sha
```

```
sha1

Cipher commands (see the `enc' command for more details)
aes-128-cbc   aes-128-ecb   aes-192-cbc   aes-192-ecb   aes-256-cbc
aes-256-ecb   base64        bf            bf-cbc        bf-cfb
bf-ecb        bf-ofb        cast          cast-cbc      cast5-cbc
cast5-cfb     cast5-ecb     cast5-ofb     des           des-cbc
des-cfb       des-ecb       des-ede       des-ede-cbc   des-ede-cfb
des-ede-ofb   des-ede3      des-ede3-cbc  des-ede3-cfb  des-ede3-ofb
des-ofb       des3          desx          idea          idea-cbc
idea-cfb      idea-ecb      idea-ofb      rc2           rc2-40-cbc
rc2-64-cbc    rc2-cbc       rc2-cfb       rc2-ecb       rc2-ofb
rc4           rc4-40
```

Wenn Sie die Liste der Befehle, die OpenSSL ausdruckt sehen, ist die Installation erfolgreich verlaufen.

Erstellen Sie zunächst einige Verzeichnisse unter *c:\OpenSSL*. Diese Verzeichnisse werden für die Zertifikate benötigt.

- *infoCA* (Rootverzeichnis für unsere Zertifizierungsstelle (**CA**))
- *infoCA\certs* (Hier kommen die Zertifikate rein)
- *infoCA\private* (Hier landet der private Schlüssel der CA)
- *infoCA\newcerts* (Hier kommen neuerstellte Zertifikate rein)
- *infoCA\user* (Hierher kommen Zertifikatsignierungsanfragen, Zertifikate usw)

Erstellen Sie die Textdatei **serial** (ohne Extension!) im Verzeichnis **infoCA** und schreiben Sie *01* hinein. Das ist die Seriennummer des nächsten Zertifikats, das erstellt wird.
Erstellen Sie im gleichen Verzeichnis die leere Textdatei **index.txt**

In den oben erzeugten Pfaden werden Zertifikate und Schlüssel gespeichert. Der Basispfad (hier *infoCA, das wird der Name unserer Zertifizierungsstelle*) kann einen beliebigen Namen haben, die Unterverzeichnisse sollten so heißen, wie hier dargestellt. Sie müssen nur nachher darauf achten, den richtigen Pfad in die Konfigurationsdatei (*openssl.cnf*) einzutragen, was wir im nächsten Abschnitt machen werden. Die beiden Dateien **serial** und **index.txt** enthalten Informationen über bereits generierte Zertifikate. Sie werden sich den Inhalt dieser Dateien später im laufenden CA-Betrieb noch einmal ansehen. Das Verzeichnis *infoCA\user* dient als Verzeichnis für die von der CA erstellten Benutzerzertifikate.

2. Erzeugen der CA *infoCA*

Der nächste Schritt ist eine Anpassung der OpenSSL-Konfigurationsdatei **c:\OpenSSL\bin\openssl.cnf.** Öffnen Sie zur Bearbeitung dieser Datei eine DOS-Box. Wechseln Sie ins Verzeichnis *c:\OpenSSL\bin* und geben Sie folgenden Befehl ein:

```
c:\OpenSSL\bin>notepad openssl.cnf
```

Die Datei **openssl.cnf** ist in mehrere Sektionen aufgeteilt. Dadurch könnte man mehrere Zertifikatsstellen auf einem Rechner verwalten. Im Beispiel benennen wir einfach die Default-CA Sektion um und verwenden sie für unsere CA *infoCA*. Wichtige Änderungen betreffen die Zeilen „dir“ (den Pfad der CA angeben).

Ändern Sie die Beispielkonfiguration folgendermaßen um (die dunkel markierten Stellen müssen verändert werden):

...

```
##################################################################
[ ca ]

default_ca = infoCA # The default ca section
##################################################################

...

[ infoCA ]
dir = c:/OpenSSL/infoCA # Where everything is kept for this CA

...

[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = DE
countryName_min = 2
countryName_max = 2
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = NRW
localityName = Locality Name (eg, city)
0.organizationName = Organization Name (eg, company)
0.organizationName_default = FH Aachen
...
organizationalUnitName          = Organizational Unit Name (eg, section)
organizationalUnitName_default  = ETIT

...
```

Hinweis: Achten Sie bei der Einstellung des Verzeichnisses der infoCA darauf, dass nicht die in Windows üblichen Rückwärtsschrägstriche, sondern normale Schrägstriche verwendet werden!

Speichern Sie die veränderte Datei und schließen sie **notepad**.

Durch diese Einstellungen werden Werte eingestellt, die Sie sonst bei jedem Programmdurchlauf von Hand eingeben müssten, Sie werden es später noch sehen.

3. Erzeugen von RSA Private und Public Keys

Dieser Abschnitt beschreibt:

- Was ist RSA?
- Erzeugung von RSA Public und Private Keys.
- Anzeigen der Komponenten eines RSA Keys.

- Verschlüsseln von RSA Keys.

Was ist RSA?

RSA ist ein asymmetrischer Verschlüsselungsalgorithmus. Er wurde 1977 entwickelt und verwendet private und öffentliche Schlüssel. RSA sind die Initialen der Entwickler des RSA Algorithmusses: Ron **R**ivest, Adi **S**hamir und Leonard **A**dleman. Heute ist RSA der im Internet am mit häufigsten verwendete Verschlüsselungsalgorithmus.

Erzeugen eines RSA Schlüsselpaares

Im Folgenden wird gezeigt, wie Sie mit **OpenSSL** einen privaten und einen öffentlichen RSA Schlüssel erzeugen können. Da es ein Schlüssel für einen Benutzer ist, soll er im Benutzerverzeichnis *c:\OpenSSL\infoSA\user* gespeichert werden. Öffnen Sie eine DOS-Box und wechseln Sie ins Verzeichnis *c:\OpenSSL\infoCA\user*.

Zur Erzeugung eines privaten Schlüssels für den Benutzer Höfken geben Sie den unten gezeigten Befehl ein.

Da hier keine Schlüssellänge angegeben ist, wird die Standardlänge von 512 verwendet. ***Diese Länge gilt heute als unsicher und wird nur aus Zeitgründen hier verwendet!***

Zeigen Sie nach dem Erzeugen den Schlüssel einmal an (er befindet sich in der Datei **hoefken_rsa.key)**.

```
c:\OpenSSL\infoCA\user> openssl genrsa -out hoefken_rsa.key
Loading 'screen' into random state - done
Generating RSA private key, 512 bit long modulus
..++++++++++++
.............++++++++++++
e is 65537 (0x10001)

c:\OpenSSL\infoCA\user> type hoefken_rsa.key
-----BEGIN RSA PRIVATE KEY-----
MIIBOwIBAAJBANoK3b+0NV1xrFLjsKFPLrxMReu3ezPxLjDWjktakq9gDGM5WUvI
CSENry/M1h2AhxGSxQluy4b1ynzBGWeO320CAwEAAQJAbQQn0NSKJflSvnLG+i/7
3vuHrg4j1FmOza5IoNZdJr9DyESMC+prebZkAFM2EW+ZLZy2JiEIqdDz79VAVRzs
ZQIhAPBvHYEWxCIcSYn8aG7o2lyY5/EB1gvwgAfSdWFlemUDAiEA6CijmKOX1WRd
KPf9g52Tpxk4TZzdjIpcbvYR7znIZs8CIBDxI3kXK5bju2LXwFwgWFKyC5X19Sk+
NydV8yN7zRYVAiEAni7CeUhONfmyeC2wsLL3Xg2TDV7qnc3QeVJ0mdl3MIUCIQCo
o0AdFXm789FfHuB+mVIKNtBLTAQNaMuXz6lXl7Ib7Q==
-----END RSA PRIVATE KEY-----
```

Hinweis:

- Die Schlüsseldatei wird im Format **PEM** erzeugt (auch wenn die Dateierweiterung **.key** lautet!).

Anzeigen der RSA Schlüsselkomponenten

So können Sie die Schlüsselkomponenten des privaten Schlüssels anzeigen:

```
c:\OpenSSL\infoCA\user> openssl rsa -in hoefken_rsa.key -text
```

```
Private-Key: (512 bit)
modulus:
    00:da:0a:dd:bf:b4:35:5d:71:ac:52:e3:b0:a1:4f:
    2e:bc:4c:45:eb:b7:7b:33:f1:2e:30:d6:8e:4b:5a:
    92:af:60:0c:63:39:59:4b:c8:09:21:0d:af:2f:cc:
    d6:1d:80:87:11:92:c5:09:6e:cb:86:f5:ca:7c:c1:
    19:67:8e:df:6d
publicExponent: 65537 (0x10001)
privateExponent:
    6d:04:27:d0:d4:8a:25:f9:52:be:72:c6:fa:2f:fb:
    de:fb:87:ae:0e:23:d4:59:8e:cd:ae:48:a0:d6:5d:
    26:bf:43:c8:44:8c:0b:ea:6b:79:b6:64:00:53:36:
    11:6f:99:2d:9c:b6:26:21:08:a9:d0:f3:ef:d5:40:
    55:1c:ec:65
Openssl prime1:
    00:f0:6f:1d:81:16:c4:22:1c:49:89:fc:68:6e:e8:
    da:5c:98:e7:f1:01:d6:0b:f0:80:07:d2:75:61:65:
    7a:65:03
prime2:
    00:e8:28:a3:98:a3:97:d5:64:5d:28:f7:fd:83:9d:
    93:a7:19:38:4d:9c:dd:8c:8a:5c:6e:f6:11:ef:39:
    c8:66:cf
exponent1:
    10:f1:23:79:17:2b:96:e3:bb:62:d7:c0:5c:20:58:
    52:b2:0b:95:f5:f5:29:3e:37:27:55:f3:23:7b:cd:
    16:15
exponent2:
    00:9e:2e:c2:79:48:4e:35:f9:b2:78:2d:b0:b0:b2:
    f7:5e:0d:93:0d:5e:ea:9d:cd:d0:79:52:74:99:d9:
    77:30:85
coefficient:
    00:a8:a3:40:1d:15:79:bb:f3:d1:5f:1e:e0:7e:99:
    52:0a:36:d0:4b:4c:04:0d:68:cb:97:cf:a9:57:97:
    b2:1b:ed
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIIBOwIBAAJBANoK3b+0NV1xrFLjsKFPLrxMReu3ezPxLjDWjktakq9gDGM5WUvI
CSENry/M1h2AhxGSxQluy4b1ynzBGWeO320CAwEAAQJAbQQn0NSKJflSvnLG+i/7
3vuHrg4j1FmOza5IoNZdJr9DyESMC+prebZkAFM2EW+ZLZy2JiEIqdDz79VAVRzs
ZQIhAPBvHYEWxCIcSYn8aG7o2lyY5/EB1gvwgAfSdWFlemUDAiEA6CijmKOX1WRd
KPf9g52Tpxk4TZzdjIpcbvYR7znIZs8CIBDxI3kXK5bju2LXwFwgWFKyC5X19Sk+
NydV8yN7zRYVAiEAni7CeUhONfmyeC2wsLL3Xg2TDV7qnc3QeVJ0mdl3MIUCIQCo
o0AdFXm789FfHuB+mVIKNtBLTAQNaMuXz6lXl7Ib7Q==
-----END RSA PRIVATE KEY-----
```

Hinweis:
Sie können die einzelnen Komponenten (Primezahl, Koeffizient usw.) identifizieren, die zur Generierung des privaten Schlüssels verwendet wurden. Am Ende sehen Sie ihren privaten Schlüssel.

Verschlüsseln der RSA Keys

Die RSA Schlüsseldatei beinhaltet den Private Key, der niemand anderem als Ihnen bekannt sein sollte. Daher wird empfohlen, sie in verschlüsselter Form (Passwort geschützt) abzuspeichern.

Im Folgenden wird gezeigt, wie Sie eine neue RSA Schlüsseldatei, verschlüsselt mit dem DES3 Algorithmus, erzeugen. Hier wird eine Schlüssellänge von 2048 Bit gewählt. Verwenden Sie als Pass Phrase hier im Praktikum immer die Zusammensetzung ***name****key* also zum Beispiel für den Benutzer Hoefken: Name=hoefken -> Pass Phrase: *hoefkenkey*. Sie verwenden Ihren eigenen Namen für die Schlüsseldatei und die Pass Phrase!

Schauen Sie sich anschließend die Schlüsseldatei wieder an.

```
c:\OpenSSL\infoCA\user> openssl genrsa -des3 -out hoefken_rsa_des.key 2048
Loading 'screen' into random state - done
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
Enter pass phrase for hoefken_rsa_des.key: hoefkenkey
Verifying - Enter pass phrase for hoefken_rsa_des.key: hoefkenkey

c:\OpenSSL\infoCA\user> type hoefken_rsa_des.key

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,66806C328A980AD2

BU7bueVKj9l+T8hE0BLAmeSmEzlv/pqhLrzob/EXUD92hnfGQ+2I5/06Z0WOiF01
zN4hVvQW+8uuwyVuzKRJOJfccODVGqE4B3ycP5EZIejrP2ccFm39pkyTQSRj9S7v
F7LfvZw0Z7xyBHt4TYyR+1JGqVl/NDn/FEcMCYEFP4j02KNISKZnZXfGgAs+dECm
Yb/gc+6o6qbDj+LhG3JmlcNSTn0SI+UHBdC/iWpIbAwHy2xI5QOO+tHfrkJS191G
2JliCGPcYKp1yvV7D54kYi+HQek09SeSI2yrgVbl3cZYoQEc5pXB2RJW4Omnw/o4
iz2E+M0aaXKw5an2tUynSkEtTnYTbzQRCX3R6FfWzqcOtTUpGL13+TKYDMmRztZ9
51AUZUQiAMDNbXAfDVFraDOz9ZZJnRYOFEolrdXrOrqQqlg9QcAQU9b9A699itHb
zreVopZ8dHKJ/mUhmSQFF4BTd//kepVlQBf6EB8CLx0jTnLcVQjAau2X2U00TIyN
5lqsMpyNKo+k+uSM8eU02XGuCo34xjvenDQPRmF06kskwUBkDnIsZ2Ymscg0kytN
P3faxDC8VEcedh3kYVLXtwtaTGO0xPVyJcQNl8zCuJAzfV5rk0wB1rXBFmSaZKTf
oKh6ODgxlwOa4SwMzAL/vRjODV0BvfFYbSlQh1h4BhOHrc375LkYHlR+cH/kG7FI
P8iXmwr1Zzmif9KqcX0Hdifdo4+yLO0Tv5pEeXDZ7I1Q7QsDaz83/Uqe/1Aqf6E9
QOF/d0gkFnHh6jtNDzGZNeKgEFFo2gk++rV7NbmDDmeJ8vzjb5vR1t9d8lzCXl7n
VT7GIIUeP9ZBfvJUs8SanS7ICrxBwEClFuAML8jHdQ1NLbrPDY1HvtFm7GQ8lZU6
4tgmwROjkzPldJu49MA8Xu2jg2yybM0+wRpPrHzF7RuRddQanICAgrlTogFZlIV4
dy+GT/0Ox+D2YldykRJ524M2RqFAj4S7R+UzoR46nWW1iKnpHHToG+cjg4f72abP
kZJcAspfnyLj/ygYqXmNFuphuag8g8LcX5CNE+Q+y4UwMID2luLpyrPpw86DUpEF
87VIDAndc/huvxcn80RKqhkD8qIVODqcAJHW5b9XgocH3WDl2IvX8fUu8oc/N/Fp
PHwohSOywNKQKuOd84KTxafQUIpxigyh4oYJf5VknyPagwpcSbe9qTrnJNlRdeL9
JkBLRU33/zA8+OffZPkN2eoIcE0hzC0qlVbLO4wM068Q5KkeOhzk5ALPgNjztfgN
TDrGGhKFIAg0/5xJjPg4qR1FTubsDZ/ruZikHugmcs6M7+L9mHcIrztUJs5idRBs
jfQQ6ozEzMC4nkESTPWWBkVMVYqBn/So1uMhU2/2CeXkcqwQ2wJlAuY7eeajhF8m
Ml5TYl7vqinJQ8H9+b2zQYhxD9ZcbdI6XIGb2sdj6xpf1KjCNWsjpfRykBBUA2v/
VSTSrSg4d8O8MbY3GvryFjgin7nhnDVeXfm4/AjLwrm9hr55gZRw22N5soUYp9qO
H3ZSJJR4o1PGseDuMdbjvF8sD3xrgCFny0nek8VxdGYJ78Qq1PLVsg==
-----END RSA PRIVATE KEY-----
```

Perfekt! Der Extraparameter **-des3** veranlasst den **genrsa** Befehl, den Schlüssel zu erzeugen und die Datei direkt mit dem DES-EDE3-CBC Algorithmus zu verschlüsseln.

Dieser Befehl kann bei der Erzeugung des privaten Schlüssels verwendet werden. Einen zugehörigen öffentlichen Schlüssel erzeugen Sie mit dem **rsa** Befehl. Dieser Befehl kann auch verwendet werden, wenn die Schlüssel manipuliert oder angezeigt werden sollen.

Mit dem Befehl **rsa** kann man Schlüssel(dateien) bearbeiten. Wie z.B. das Verschlüsseln einer existierenden Schlüsseldatei funktioniert, führen wir jetzt einmal durch. Da Sie eine verschlüsselte Schlüsseldatei verwenden, müssen Sie deren Pass Phrase eingeben (s.o.)!

```
c:\OpenSSL\infoCA\user> openssl rsa -in hoefken_rsa.key -des -out hoefken_rsa_des.key
writing RSA key
Enter PEM pass phrase: hoefkenkey
Verifying - Enter PEM pass phrase: hoefkenkey

c:\OpenSSL\infoCA\user> type hoefken_rsa_des.key

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-CBC,C386818044590B20

10JtM182aDIEMcGLGHXC51woLVdcsYWAAp0tCI1NKJRy/ZBKQLs7gzgGD9ZFBA3D
eZ0W7CVT226yDNSAq/3G+st1cR3kfFmxO3cfT8DHKV4zJVLSRrKfklURp0SdfaB6
LLpbdz9OSwxYphVTBTQAaeLYBipZhyV5BJZeQH40b5S3SclHid5Bn3SaxmFIgRCp
X07GQkiVU+KLhW4Q2v7uV7qU/dlym7WAsxlw4vEw9EhLw2RTPGEC0IaTzPtgWnsE
wQcvS0gDg5C8sP/rpHdQcZFCqpt4+n9M/p1Ciz1d0DNYRefvZnmf9w/z02oT3KY+
nJxrL6kh2kYVUOQKSwlA4Swtt4lPy6gimg+1xG96+BnrG803FYQ23rlusCThg+yw
lHpltupnF9YW38dParIlLsxMxFRhRc8qNZSAwnBHP78=
-----END RSA PRIVATE KEY-----
```

Wie sie sehen, ist die Datei nun im Klartext nicht mehr lesbar!

Als nächstes werden wir aus der eben erzeugten Datei (mit dem privaten Schlüssel) den Public Key erzeugen (Parameter *pubout*). Da die Schlüsseldatei verschlüsselt ist, müssen wir bei dieser Aktion wieder unsere Pass Phrase eingeben. Auch das wird mit dem Befehl **rsa** durchgeführt.

```
c:\OpenSSL\infoCA\user> openssl rsa -in hoefken_rsa_des.key -pubout -out hoefken_rsa_des_pub.key
Enter pass phrase for hoefken_rsa_des.key: hoefkenkey
Writing RSA key

c:\OpenSSL\infoCA\user> type hoefken_rsa_des_pub.key

-----BEGIN PUBLIC KEY-----
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAMQxSqkk+wvk+UrnwFZqWX15ioURqrzT
1yMCg4abdgVsHvdDISobaLx9yD3Z0dJF1rr6+lo1P6fC/wk5GgF+AMUCAwEAAQ==
-----END PUBLIC KEY-----
```

Nach diesem Befehl liegt der Public Key in der Datei **c:\OpenSSL\infoCA\user\hoefken_rsa_des_pub.key**.

Die Datei **c:\OpenSSL\infoCA\user\hoefken_rsa_des.key** ist bei dieser Aktion nicht verändert worden!

Zusammenfassung

Sie haben zwei Befehle kennengelernt: ***genrsa*** und ***rsa***. Sie können zur Erzeugung, Verschlüsselung und Veränderung von RSA Schlüsseln verwendet werden.

4. Erzeugen des CA Root-Zertifikats

Der nächste Schritt öffnet jetzt endgültig die Tür zum Betrieb einer CA. Es wird das Schlüsselpaar für die CA erstellt, und ein ROOT Zertifikat, das selbst signiert ist (d.h. man selber beglaubigt, dass das Zertifikat korrekt ist :-). Mit diesem Zertifikat sind Sie dann in der Lage, andere Zertifikate auszustellen und zu signieren.

Verwenden Sie als Pass-Phrase: **INFO2012**
Geben Sie unten die entsprechenden Daten ein (einige haben wir schon in die **openssl.cnf** eingegeben und Sie brauchen sie nur noch mit **Return** bestätigen!):

Country Name: *DE*
State or Province: *NRW*
Locality Name: *Aachen*
Organization Name: *FH Aachen*
Organizaional Unit Name: *INFO*
Common Name: *CA*
Email Address: *ca@fh-aachen.de*

Das Zertifikat soll 10 Jahre gültig sein.

Geben Sie zuerst den folgenden Befehl ein und schauen Sie sich dann das Ergebnis an:

```
c:\OpenSSL\infoCA\user > openssl req -new -x509 -keyout ..\private\cakey.pem -out ..\cacert.pem -days 3650
Loading 'screen' into random state - done
Generating a 1024 bit RSA private key
.....++++++
.....++++++
writing new private key to '..\private\cakey.pem'
Enter PEM pass phrase: INFO2012
Verifying - Enter PEM pass phrase: INFO2012
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [DE]: (return)
State or Province Name (full name) [NRW]: (return)
Locality Name (eg, city) []:Aachen
Organization Name (eg, company) [FH Aachen]: (return)
Organizational Unit Name (eg, section) []:INFO
Common Name (eg, YOUR name) []:CA
Email Address []:ca@fh-aachen.de
```

```
c:\OpenSSL\infoCA\user > type ..\private\cakey.pem
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,49F02F7BBD8035AF

W485wks+lAdcFhJOUOb7ywT9HEQmVKOwubFMp925NrtCvHZd04PEIq349TmYSH+a
SbXuFEb6eUzI/mBYCjvV4OeEBktFqbdYYJj4E/gv5nhUnLQvBRMS6KTv23g3ESIe
H1TjI9OjZrSnNNB4O+fOdsoUJc+c5MjnnqmJ4ym3gidNxt6ws3gJY338PQlgAP4Z
8e/zn8ajaWF4YpBypd7AyH32/GSFvY2YxN2P9ToXEzr8QL3ondmvhsQuNTtwuUI
ELuxo3JvQ4XsZOTqgqlA2OefJWsHc/unWERwqsTLno/qs3pMR+3Z6c+bdGJSCoVe
pmfVo4UYr89A5lkHXmP3h+ZRbwE6Id/oYiB58XJglB89ncsTXXi+om0qTGco+Zyx
hRWsCQMsGoV6K7uDUiodyjsvyjA+i3js8NyJSiK6lx6fcQb4QBz5KpqchHtPPcfS
2XqXImjFUWEQDfKmuxuc+1TNsCvbQu8uqiP5Tfh1XO/10GX8iU5olJeKg8qL1f+u
hw3p82tyXbX2g22Tfo0CE5Kl86ZcVE71+KIMw5DEhaFr7i8V/czydPyE22txXEyG
KxO4U24Ec0h75HQNE1UjaFwZbAA4PDBWh2uW0WeQY5+N7zou2at2avFrthDrhYcR
3YPb1h9x3BncU/eMOT7X4pWHcHmkJEwbOJWMmhBah3MGViKVd83L2QvDtoWwaz5u
8n+MAE8YJim4wMuSvQM/V0spUYLXJ8ILy7f7F45n1Ub6mnxNgIMBUtUTh3f+7gxs
MNNrE1FSJ+x96565zCn41bhU41fQJYn895fbkt8/e6uzrbZ4dY0Edg==
-----END RSA PRIVATE KEY-----

c:\OpenSSL\infoCA\user > type ..\cacert.pem
-----BEGIN CERTIFICATE-----
MIIDVTCCAr6gAwIBAgIJAJLlN3HL7RuSMA0GCSqGSIb3DQEBBQUAMHsxCzAJBgNV
BAYTAkRFMQwwCgYDVQQIEwNOUlcxDzANBgNVBAcTBkFhY2hlbjESMBAGA1UEChMJ
RkggQWFjaGVuMQwwCgYDVQQLEwNTRE4xCzAJBgNVBAMTAkNBMR4wHAYJKoZIhvcN
AQkBFg9jYUBmaC1hYWNoZW4uZGUwHhcNMDgwMzI1MTUxNzI0WhcNMTgwMzIzMTUx
NzI0WjB7MQswCQYDVQQGEwJERTEMMAoGA1UECBMDTlJXMQ8wDQYDVQQHEwZBYWNo
ZW4xEjAQBgNVBAoTCUZIIEFhY2hlbjEMMAoGA1UECxMDU0ROMQswCQYDVQQDEwJD
QTEeMBwGCSqGSIb3DQEJARYPY2FAZmgtYWFjaGVuLmRlMIGfMA0GCSqGSIb3DQEB
AQUAA4GNADCBiQKBgQCkQCXOfZD01i/FinPqSuH2sfebegdsqrRoso7pjWHI5xn0
xrp3FXY1i4IJfPBne8bYN+v6MVdH2U6kqugng9mLSv9zM/NRHXn03dqK0RSboWxd
i7OwPNaGSHfHuPCFAUuCM8cA4H2hKFtSQiz3MRu8s2KWwckbB4Y8xyhYkE9oOwID
AQABo4HgMIHdMB0GA1UdDgQWBBSbTpE+cAFMZQAX/uYVBipDkkUAkTCBrQYDVR0j
BIGlMIGigBSbTpE+cAFMZQAX/uYVBipDkkUAkaF/pH0wezELMAkGA1UEBhMCREUx
DDAKBgNVBAgTA05SVzEPMA0GA1UEBxMGQWFjaGVuMRIwEAYDVQQKEwlGSCBBYWNo
ZW4xDDAKBgNVBAsTA1NETjELMAkGA1UEAxMCQ0ExHjAcBgkqhkiG9w0BCQEWD2Nh
QGZoLWFhY2hlbi5kZYIJAJLlN3HL7RuSMAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcN
AQEFBQADgYEALh2xeOF9cNkBH2qItifEGY1jkGlGjsLO20Goql/2Y1hhV04evKwe
dVcdC5g+bBGymmkUQFXn1OE1a7d1Q/kq1b/F45SzGzGGpSY6jDcuUBXv1RZLguoY
tAqYwvfuYjByVhm2cUt6+Lm9u71/fQEovRGQ4Nm529ClLzXu3aCixFs=
-----END CERTIFICATE-----
```

Hinweis:
Dieser Befehl erzeugt zwei Dateien. Zum einen die Datei **c:\OpenSSL\infoCA\private\cakey.pem**, die den Private Key der CA enthält. Dieser ist **unter allen Umständen** zu schützen. Wenn dieser Key in fremde Hände gerät, ist das gesamte Zertifikatssystem nutzlos und alle ausgestellten Zertifikate sind ungültig!

Die zweite erstellte Datei (**c:\OpenSSL\infoCA\cacert.pem**) beinhaltet das eigentliche Zertifikat. Es ist 10 Jahre gültig (kann man per Parameter *-days* einstellen). Den Inhalt des Zertifikats können Sie sich mit dem Befehl **openssl x509 -in ..\cacert.pem -noout –text** anzeigen lassen (tun Sie das jetzt!).

Von wann bis wann ist das Zertifikat gültig (genaues Datum)? **5. Nov.2020**

Welche Schlüssellänge hat der RSA Schlüssel? **1024 Bit**

Welchen Schlüssel beinhaltet das Zertifikat? **Den öffentlichen Schlüssel der CA**

Damit ist die Zertifizierungsstelle fertig und wir können beginnen, unsere eigenen Zertifikate auszustellen.

5. Erzeugung von Selbstsignierten (Self-Signed) Zertifikaten

Dieser Abschnitt beschreibt:

- Was ist ein Zertifikat?
- Erzeugung von selbstsignierten Zertifikaten
- Anzeigen der Komponenten eines Zertifikates

Was ist ein Zertifikat?

Zertifikat: Eine vom Zertifikatherausgeber digital *signierte Aussage*, die bestätigt, dass der *öffentliche Schlüssel* des *Zertifikatbesitzers,* der im Zertifikat enthalten ist, einen bestimmten Wert hat.

Diese Definition ist vom JDK1.3.1 kopiert. Es trifft einige wichtige Aussagen:

i. "signierte Aussage" – Das Zertifikat muss vom Herausgeber mit einer digitalen Signatur signiert sein.
ii. "Zertifikatherausgeber“ – Die Person oder Organisation, die dieses Zertifikat erzeugt bzw. herausgegeben hat.
iii. "öffentlicher Schlüssel" – Der öffentliche Schlüssel eines Schlüsselpaares
iv. "Zertifikatbesitzer" – Die Person oder Organisation, die den öffentlichen Schlüssel besitzt (und sinnvollerweise den passenden privaten Schlüssel natürlich auch!).

X.509 Zertifikat – Ein Zertifikat im Format X.509. Der X.509 Standard wurde 1988 eingeführt. Er beschreibt ein Zertifikat, das folgende Informationen beinhalten muss:

- Version - X.509 Standard Versionsnummer.
- Seriennummer – Eine Zahlenfolge, die jedem Zertifikat gegeben wird.
- Verwendeter Signaturalgorithmus - Name des Algorithmus, die der Herausgeber zum Signieren dieses Zertifikats verwendet hat.
- Name des Herausgebers
- Gültigkeitsperiode – Periode, während der das Zertifikat gültig ist.
- Subjekt Name - Name des Besitzers des öffentlichen Schlüssels.
- Subject Public Key Information – Der öffentliche Schlüssel und seine zugehörigen Informationen.

Erzeugen von selbstsignierten Zertifikaten

Ein selbstsigniertes Zertifikat ist ein Zertifikat, bei dem der **Herausgeber** gleichzeitig der **Zertifikatsbesitzer** ist. Mit anderen Worten: der Herausgeber signiert seinen eigenen öffentlichen Schlüssel mit seinem privaten Schlüssel.

Um so ein selbstsigniertes Zertifikat zu erstellen müssen Sie Folgendes tun:

- Den eigenen Namen als *subject* eingeben.
- Zufügen des eigenen öffentlichen Schlüssels.

- Signieren mit dem eigenen privaten Schlüssel.
- Alles im X.509 Format zusammenfügen.

Das hört sich nach viel Arbeit an, aber **OpenSSL** erledigt das alles mit dem ***req*** Befehl auf einen Schlag für Sie.

Hier nun der Befehl, um ein selbstsigniertes Zertifikat, basierend auf der RSA Schlüsseldatei **hoefken_rsa_des.key** mit dem privaten Schlüssel der Benutzers Höfken, die wir vorher schon erstellt haben, zu erzeugen:

```
c:\OpenSSL\infoCA\user> openssl req -new -key hoefken_rsa_des.key -x509 -out hoefken.crt

Enter pass phrase for hoefken_rsa_des.key: hoefkenkey
You are about to be asked to enter information that will be incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) DE[]: (return)
State or Province Name (full name) [NRW]: (return)
Locality Name (eg, city) []:Aachen
Organization Name (eg, company) [FH Aachen]: (return)
Organizational Unit Name (eg, section) []:ETIT
Common Name (eg, YOUR name) []:Hans Hoefken
Email Address []:hoefken@fh-aachen.de

c:\OpenSSL\sdnCA\user> type hoefken.crt
-----BEGIN CERTIFICATE-----
MIICUTCCAfugAwIBAgIBADANBgkqhkiG9w0BAQQFADBXMQswCQYDVQQGEwJDTjEL
MAkGA1UECBMCUE4xCzAJBgNVBAcTAkNOMQswCQYDVQQKEwJPTjELMAkGA1UECxMC
VU4xFDASBgNVBAMTC0hlcm9uZyBZYW5nMB4XDTA1MDcxNTIxMTk0N1oXDTA1MDgx
NDIxMTk0N1owVzELMAkGA1UEBhMCQ04xCzAJBgNVBAgTAlBOMQswCQYDVQQHEwJD
TjELMAkGA1UEChMCT04xCzAJBgNVBAsTAlVOMRQwEgYDVQQDEwtIZXJvbmcgWWFu
ZzBcMA0GCSqGSIb3DQEBAQUAA0sAMEgCQQCp5hnG7ogBhtlynpOS21cBewKE/B7j
V14qeyslnr26xZUsSVko36ZnhiaO/zbMOoRcKK9vEcgMtcLFuQTWDl3RAgMBAAGj
gbEwga4wHQYDVR0OBBYEFFXI70krXeQDxZgbaCQoR4jUDncEMH8GA1UdIwR4MHaA
FFXI70krXeQDxZgbaCQoR4jUDncEoVukWTBXMQswCQYDVQQGEwJDTjELMAkGA1UE
CBMCUE4xCzAJBgNVBAcTAkNOMQswCQYDVQQKEwJPTjELMAkGA1UECxMCVU4xFDAS
BgNVBAMTC0hlcm9uZyBZYW5nggEAMAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcNAQEE
BQADQQA/ugzBrjjK9jcWnDVfGHlk3icNRq0oV7Ri32z/+HQX67aRfgZu7KWdI+Ju
Wm7DCfrPNGVwFWUQOmsPue9rZBgO
-----END CERTIFICATE-----
```

Hinweis:

- Der öffentliche Schlüssel, der in das Zertifikat eingefügt wird, wird aus dem privaten Schlüssel, der in der RSA Schlüsseldatei **hoefken_rsa_des.key** ist erstellt.
- Der private Schlüssel, mit dem das Zertifikat signiert wird, wird ebenfalls der RSA Schlüsseldatei **hoefken_rsa_des.key** entnommen. Der private Schlüssel wird **nicht** zum Zertifikat hinzugefügt. Daher kann dieses Zertifikat auch an andere versendet werden!
- Das Zertifikat wird im PEM Format gespeichert.

Anzeigen der Zertifikatskomponenten

So können Sie die Komponenten des Zertifikats anzeigen:

```
c:\OpenSSL\infoCA\user> openssl x509 -in hoefken.crt -noout -text
Certificate:
 Data:
  Version: 3 (0x2)
  Serial Number: 0 (0x0)
  Signature Algorithm: md5WithRSAEncryption
  Issuer: C=DE, ST=NRW, L=Aachen, O=FH Aachen, OU=ETIT, CN=Hans
     Hoefken/emailAddress=hoefken@fh-aachen.de
  Validity
   Not Before: Jul 15 02:19:47 2002 GMT
   Not After : Aug 14 02:19:47 2002 GMT
  Subject: C=DE, ST=NRW, L=Aachen, O=FH Aachen, OU=ETIT, CN=Hans
     Hoefken/emailAddress=hoefken@fh-aachen.de
  Subject Public Key Info:
   Public Key Algorithm: rsaEncryption
   RSA Public Key: (512 bit)
    Modulus (512 bit):
     00:a9:e6:19:c6:ee:88:01:86:d9:72:9e:93:92:db:
     57:01:7b:02:84:fc:1e:e3:57:5e:2a:7b:2b:25:9e:
     bd:ba:c5:95:2c:49:59:28:df:a6:67:86:26:8e:ff:
     36:cc:3a:84:5c:28:af:6f:11:c8:0c:b5:c2:c5:b9:
     04:d6:0e:5d:d1
    Exponent: 65537 (0x10001)
  X509v3 extensions:
   X509v3 Subject Key Identifier:
    55:C8:EF:49:2B:5D:E4:03:C5:98:1B:68:24:28:47:88:D4:0E:77:04
   X509v3 Authority Key Identifier:
    keyid:55:C8:EF:49:2B:5D:E4:03:C5:98:1B:68:24:28:47:88:D4:0E:77:04
    DirName:/C=DE/ST=NRW/L=Aachen/O=FH Aachen/OU=ETIT/CN=Hans
       Hoefken/emailAddress=hoefken@fh-aachen.de
    serial:00
   X509v3 Basic Constraints:
    CA:TRUE
 Signature Algorithm: md5WithRSAEncryption
  3f:ba:0c:c1:ae:38:ca:f6:37:16:9c:35:5f:18:79:64:de:27:
  0d:46:ad:28:57:b4:62:df:6c:ff:f8:74:17:eb:b6:91:7e:06:
  6e:ec:a5:9d:23:e2:6e:5a:6e:c3:09:fa:cf:34:65:70:15:65:
  10:3a:6b:0f:b9:ef:6b:64:18:0e
```

Das Zertifikat gibt uns folgende Informationen:

- Das *Subjekt* (Besitzer) ist "C=DE, ST=NRW, L=Aachen, O=FH Aachen, OU=ETIT, CN=Hans Hoefken" mit der E-Mail-Adresse hoefken@fh-aachen.de
- Der öffentliche Schlüssel des Besitzers ist im Zertifikat vorhanden.
- Der *Issuer* (Herausgeber) ist "C=DE, ST=NRW, L=Aachen, O=FH Aachen, OU=ETIT, CN=Hans Hoefken“ mit der E-Mail-Adresse hoefken@fh-aachen.de. Der Herausgeber ist gleichzeitig der Besitzer (es ist ein selbstsigniertes Zertifikat).
- Das Zertifikat ist einen Monat lang gültig.

- Das Zertifikat ist vom Herausgeber mit der Signatur am Ende signiert.

Zusammenfassung

In diesem Kapitel haben wir den Befehl **req** verwendet, um ein selbstsigniertes Zertifikat zu erzeugen und dann die Zertifikatskomponenten angezeigt.

6. Signieren von Zertifikaten von Dritten

Dieser Abschnitt beschreibt:

- Warum müssen Zertifikate von CA's signiert werden?
- Erzeugen einer Zertifikatssignierungsanfrage für Ihren öffentlichen Schlüssel
- Anzeigen der Komponenten einer Zertifikatssignierungsanfrage
- Signieren einer Zertifikatssignierungsanfrage

Warum müssen Zertifikate von einer CA signiert werden?

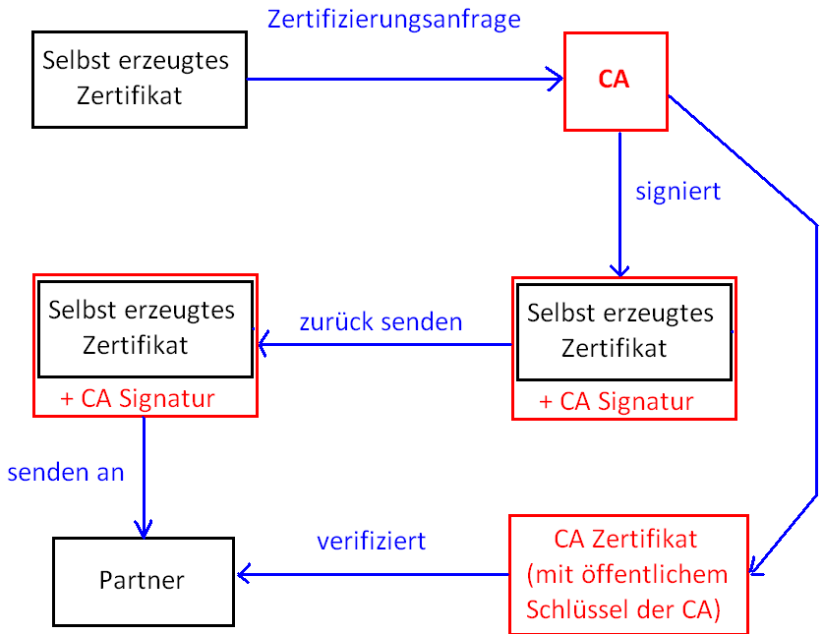
Im vorhergehenden Kapitel haben Sie gelernt, wie Sie Ihren eigenen öffentlichen Schlüssel in ein Zertifikat eingebracht und anschließend mit Ihrem eigenen privaten Schlüssel signiert haben, um ein selbstsigniertes Zertifikat zu erstellen.

Natürlich können Sie dieses Zertifikat an Ihren Kommunikationspartner senden und es zum verschlüsseln von Daten nutzen. Das funktioniert aber nur, wenn Ihr Partner Sie kennt und er Ihrer digitalen Signatur vertraut. Auch muss der Transportweg des Zertifikats sicher sein, so dass es auf dem Transport nicht verändert werden kann!

Für den Fall, dass Ihr Kommunikationspartner Ihnen nicht direkt vertraut wird es schwieriger. Sie können jetzt Ihren öffentlichen Schlüssel an eine Zertifizierungsstelle (*CA*) senden, um es dort von der CA signieren zu lassen. Dazu müssen Sie Ihren öffentlichen Schlüssel in eine Zertifikatssignierungsanfrage (certificate signing request, *CSR*) einfügen, und diese zur CA senden. Die CA wird die Anfrage überprüfen/verifizieren, Ihren öffentlichen Schlüssel in ein Zertifikat einfügen und dieses Zertifikat mit dem privaten Schlüssel der CA signieren. Dazu müssen Sie in der Regel mit einem Ausweis in der CA erscheinen und sich ausweisen.

Wenn Ihr Partner Ihren öffentlichen Schlüssel, von der CA signiert (der er vertraut!), empfängt, kann er das Zertifikat mit dem öffentlichen Schlüssel der CA überprüfen. Sollte die Prüfung positiv ausfallen, kann er Ihrem öffentlichen Schlüssel vertrauen.

Hier ein einfaches Diagramm, das den Prozess der Zertifikatsignierung und den Prüfungsprozess darstellt:



Das CA-Zertifikat enthält auch den öffentlichen Schlüssel der CA, mit dem die Signatur Ihres selbsterzeugten Zertifikats verifiziert werden kann.

Erzeugen einer Zertifikatssignierungsanfrage (Certificate Signing Request) für Ihren eigenen öffentlichen Schlüssel

Um Ihren öffentlichen Schlüssel zur Signierung an die CA zu senden, müssen Sie ihn in eine Zertifikatsignierungsanfragedatei (CSR) einfügen. Hier ein Beispiel, wie Sie das mit dem *req* Befehl machen (**Achtung:** Geben Sie dieses Mal auf ein *Challenge Passwort* ein wenn Sie dazu aufgefordert werden, nur so…). Der Benutzer Hoefken möchte nun einen selbsterstellten öffentlichen Schlüssel in einem Zertifikat, das von der CA signiert ist, haben.

Geben Sie den folgenden Befehl ein, um eine CSR zu erstellen:

```
c:\OpenSSL\infoCA\user> openssl req -new -key hoefken_rsa_des.key -out hoefken.csr

Enter pass phrase for hoefken_rsa_des.key: hoefkenkey
You are about to be asked to enter information that will be incur…
into your certificate request.
What you are about to enter is what is called a Distinguished Name...
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [DE]: (return)
State or Province Name (full name) [NRW]:(return)
Locality Name (eg, city) []:Aachen
Organization Name (eg, company) [FH Aachen]: (return)
Organizational Unit Name (eg, section) []:ETIT
Common Name (eg, YOUR name) []:Hans Hoefken
Email Address []: hoefken@fh-aachen.de

Please enter the following 'extra' attributes
to be sent with your certificate request
```

```
A challenge password []:meinreq
An optional company name []:(return)

c:\OpenSSL\infoCA\user> type hoefken.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIBETCBvAIBADBXMQswCQYDVQQGEwJDTjELMAkGA1UECBMCUE4xCzAJBgNVBAcT
AkNOMQswCQYDVQQKEwJPTjELMAkGA1UECxMCVU4xFDASBgNVBAMTC0hlcm9uZyBZ
YW5nMFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKnmGcbuiAGG2XKek5LbVwF7AoT8
HuNXXip7KyWevbrFlSxJWSjfpmeGJo7/Nsw6hFwor28RyAy1wsW5BNYOXdECAwEA
AaAAMA0GCSqGSIb3DQEBBAUAA0EALE+d7H514HyQXu2CgwXYDvqZRngFLZFdGxQN
6AtEXXV+eC2c+URNBcmoF3oghJdPqZv7D1nZ7EBf20XSWzioQA==
-----END CERTIFICATE REQUEST-----
```

Hinweis: Das Zertifikat wird im PEM Format gespeichert.

Anzeigen der Komponenten der Zertifikatsignierungsanfrage

Mit diesem Befehl können Sie die Komponenten der Zertifikatsignierungsanfrage anzeigen:

```
c:\OpenSSL\infoCA\user> openssl req -in hoefken.csr -noout -text
Certificate Request:
   Data:
      Version: 0 (0x0)
      Subject: C=DE, ST=NRW, L=Aachen, O=FH Aachen, OU=ETIT, CN=Hans Hoefken
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
         RSA Public Key: (512 bit)
            Modulus (512 bit):
               00:a9:e6:19:c6:ee:88:01:86:d9:72:9e:93:92:db:
               57:01:7b:02:84:fc:1e:e3:57:5e:2a:7b:2b:25:9e:
               bd:ba:c5:95:2c:49:59:28:df:a6:67:86:26:8e:ff:
               36:cc:3a:84:5c:28:af:6f:11:c8:0c:b5:c2:c5:b9:
               04:d6:0e:5d:d1
            Exponent: 65537 (0x10001)
      Attributes:
         challengePassword       :meinreq
   Signature Algorithm: md5WithRSAEncryption
      80:be:77:39:65:0f:24:db:70:c1:76:e3:b6:c7:99:a5:c7:af:
      ae:98:5a:73:98:f8:60:f1:65:08:a9:f7:df:6f:bd:77:aa:f7:
      bb:0b:f2:0d:71:6e:ad:ee:52:5a:2b:a7:2a:c0:fd:0e:4c:8f:
      c1:43:18:58:0b:10:03:e0:e5:a3
```

Einige interessante Anmerkungen:

- Die Anfrage ist mit dem eigenen privaten Schlüssel signiert…warum?

  Damit niemand die Anfrage verändern kann (z.B. seinen eigenen öffentlichen Schlüssel einfügen!)

- Das "challengePassword" wird in Klartext angezeigt. Welchen Wert hat ein Passwort, wenn jeder es sehen kann?

  Hat keinen Sinn, kann man ruhig weglassen. Es kann höchstens dazu verwendet werden, Verwechslungen zu vermeiden, einen Sicherheitsaspekt hat es nicht.

Signieren einer Zertifikatsignierungsanfrage

Auch wenn Sie keine etablierte CA sind, können Sie doch **OpenSSL** verwenden, um das Zertifikat eines Dritten zu signieren. Der folgende Prozess zeigt, wie das Zertifikat von Hans Hoefken mit dem privaten Schlüssel der CA signiert wird:

```
Signieren von Hans Hoefken's Anfrage mit dem privaten Schlüssel der CA

c:\OpenSSL\infoCA\user> openssl ca -in hoefken.csr -out hoefken.crt
Using configuration from C:\OpenSSL\bin\openssl.cnf
Loading 'screen' into random state - done
Enter pass phrase for c:/OpenSSL/infoCA/private/cakey.pem:INFO2012
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 2 (0x2)
        Validity
            Not Before: Mar 25 16:11:35 2008 GMT
            Not After : Mar 25 16:11:35 2009 GMT
        Subject:
            countryName               = DE
            stateOrProvinceName       = NRW
            organizationName          = FH Aachen
            organizationalUnitName    = ETIT
            commonName                = Hans Hoefken
            emailAddress              = hoefken@fh-aachen.de
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                53:FA:55:AB:5B:23:41:B3:C6:3C:C7:41:8B:5C:68:5C:7D:B2:CC:E0
            X509v3 Authority Key Identifier:
                keyid:6A:4A:44:CE:11:1D:4F:FD:AD:B2:A4:82:CE:51:0A:CD:67:35:7C:E8

Certificate is to be certified until Mar 25 16:11:35 2009 GMT (365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

Hinweis:

In der Ausgabe sehen Sie, welcher Schlüssel zur Signierung verwendet wird (**c:/OpenSSL/infoCA/private/cakey.pem**). Es ist der private Schlüssel der CA. Sie sehen die Seriennummer des Zertifikats, die Gültigkeitsdauer (1 Jahr) und für wen dieses Zertifikat ist.

Schauen Sie sich einmal das Zertifikat an.

Welche Seriennummer hat das Zertifikat? **1**

Wer ist der Herausgeber? **Issuer: ca@fh-aachen.de**

Wer ist der Besitzer? **Subject: Hans Hoefken**

Anschließend wird die Datenbank der CA auf den neuesten Stand gebracht.

Das können Sie sich auch mit einem Texteditor ansehen.

Öffnen Sie die folgende Datei und tragen Sie den aktuellen Wert unten ein:

infoCA\serial Wert: **2**

infoCA\index.txt

infoCA\newcerts\01.pem

Schön oder? Nun können Sie beliebige Zertifikate signieren, Sie können eine CA sein. Alles was Sie benötigen ist das RSA Schlüsselpaar der CA.

Es gibt mehr als ein Dateiformat für Zertifikate. Der Internet Explorer von Microsoft und auch Firefox verwenden z.B. das PKCS#12-Format. Auch hier kann **OpenSSL** helfen. Wir exportieren das Benutzer-Zertifikat nun noch in das PKCS#12-Format.

```
c:\OpenSSL\infoCA\user> openssl pkcs12 -export -in hoefken.crt -inkey hoefken_rsa_des.key -out hoefken.p12
Loading 'screen' into random state - done
Enter pass phrase for hoefken_rsa_des.key:hoefkenkey
Enter Export Password: test
Verifying - Enter Export Password: test
```

Wer paranoid veranlagt ist, kann ein Export-Passwort vergeben. Das muss aber nicht sein. Im Verzeichnis *c:\OpenSSL\infoCA\user* befinden sich jetzt folgende Dateien:

hoefken_rsa.key – erster erzeugter Schlüssel (nur für Demozwecke), kann gelöscht werden
hoefken_rsa_des.key - mein Private Key. *Diese Datei darf nicht auf dem OpenSSL-Server bleiben und muss gut geschützt werden!*
hoefken_rsa_des_pub.key - mein Public Key. Wird unter anderem auf dem OpenSSL-Server eingesetzt.
hoefken.csr - die Zertifikatsanforderung (certificate signing request). Kann gelöscht werden.
hoefken.crt - mein Zertifikat im PEM-Format.
hoefken.p12 - mein Zertifikat im PKCS#12-Format.

Sie könnten jetzt beispielsweise alle Dateien in diesem Verzeichnis dem Benutzer Hoefken übergeben, der sie dann verwendet, um Zugang zum Server zu bekommen.
Der eben beschriebene Vorgang ist recht komplex und Fehler können überall passieren. Tritt ein Problem auf, kann meistens eine Datei nicht gefunden werden. Dann muss geprüft werden, ob die

Verzeichnisse korrekt sind, ob die Datei auch dort ist, wo sie sein soll usw. In den meisten Fällen ist auch die OpenSSL-Website eine gute Hilfe.

Zusammenfassung

In diesem Abschnitt haben Sie gesehen, wie Sie eine Zertifikatsignierungsanfrage mit dem Befehl *req* erzeugen und wie Sie mit dem Befehl *CA* das Zertifikat von der CA mit ihrem privaten Schlüssel signieren können.

7. Zertifikatspfade und deren Prüfung

Dieser Abschnitt beschreibt:

- Was ist ein Zertifikatspfad?
- Zertifikatspfadprüfung
- Testen des Zertifikatspfads mit OpenSSL

Was ist ein Zertifikatspfad?

Zertifikatspfad (auch Zertifikatskette genannt): Eine geordnete Liste von Zertifikaten, bei denen der Zertifikatsbesitzer identisch mit dem Herausgeber des nächsten Zertifikats ist.

Ein Zertifikatspfad kann auch als geordnete Liste von Zertifikaten bezeichnet werden, bei denen der Herausgeber eines Zertifikats als Besitzer des vorherigen Zertifikats identifiziert werden kann. Das erste Zertifikat ist allerdings ein Besonderes, da es kein vorheriges Zertifikat gibt. Daher muss das erste Zertifikat ein selbstsigniertes Zertifikat sein (Besitzer und Herausgeber sind gleich).

Das folgende Beispiel zeigt einen Zertifikatpfad:

```
Certificate 1
  Issuer: caadmin
  Subject: caadmin

Certificate 2
  Issuer: caadmin
  Subject: Marko Schuba

Certificate 3
  Issuer: Marko Schuba
  Subject: Stefan Nagel

Certificate 4
  Issuer: Stefan Nagel
  Subject: Georg Hoever
```

Zertifikatspfadsprüfung (Certification Path Validation)

Ein Zertifikastpfad muss geprüft werden. Hier die verwendeten Regeln:

- Das erste Zertifikat muss selbstsigniert sein. Der Herausgeber muss eine vertrauenswürdige CA sein!
- Der Herausgeber jedes folgenden Zertifikats muss identisch zum Besitzer des vorherigen Zertifikats sein.

- "identisch" bedeutet, dass die Herausgebersignatur durch den öffentlichen Schlüssel des Besitzers des vorherigen Zertifikats bestätigt werden kann.

OpenSSL bietet den **verify** Befehl, um einen Zertifikatspfad zu prüfen. Hier die Syntax des "verify" Befehls:

*verify -CAfile **first.crt** -untrusted **all_middle.crt last.crt***

- "first.crt" ist das erste Zertifikat des Pfads, ein selbstsigniertes Zertifikat.
- "last.crt" ist das letzte Zertifikat des Pfads.
- "all_middle.crt" ist eine Anzahl von mittleren Zertifikaten. Falls die Zertifikate im PEM Format gespeichert sind, können Sie sie in einem normalen Texteditor aneinanderfügen (das machen wir später auch so!).

Testen eines Zertifikatpfads mit OpenSSL

Hier nun ein Testszenario, in dem der Pfad mehrerer Zertifikate mit unterschiedlichen Herausgebern und Besitzern geprüft wird.

a. Erzeugen eines selbstsignierten Zertifikats für CA-Admin: caadmin.crt:

Verwenden Sie *caadminkey* als Pass Phrase und alle weiteren einzugebenden Werte wie bisher gezeigt!

Erzeugen der Schlüssel von CA-Admin
c:\OpenSSL\infoCA\user> **openssl genrsa -des3 -out caadmin_rsa.key**
...

Erzeugen eines selbstsignierten Zertifikats für CA-Admin
c:\OpenSSL\infoCA\user> **openssl req -new -key caadmin_rsa.key -x509 -out caadmin.crt**
...

b. Erzeugen eines Zertifikats für Marko Schuba und Signierung durch CA-Admin.

Ergebnis: schuba.crt:

Erzeugen der Schlüssel von Marko Schuba
c:\OpenSSL\infoCA\user> **openssl genrsa -des3 -out schuba_rsa.key**
...

Erzeugen der Zertifikatsignierunganforderung für Marko Schuba
c:\OpenSSL\infoCA\user> **openssl req -new -key schuba_rsa.key -out schuba.csr**
...

Signierung von Marko Schuba's Anfrage durch den Schlüssel des caadmin's.
C:\OpenSSL\infoCA\user> **openssl x509 -req -in schuba.csr -extfile c:\openssl\bin\openssl.cnf -extensions v3_ca -CA caadmin.crt -CAkey caadmin_rsa.key -out schuba.crt -set_serial 3**
...

Hinweis: Die Angabe der Parameter **-extfile** und **extensions** muss bei der Signierung verwendet werden, damit die erstellten Zertifikate berechtigt sind, weitere Zertifikate zu signieren, ansonsten gäbe es bei der Zertifikatskettenüberprüfung (s.u.) einen Fehler. Schauen Sie sich in der Datei **c:\OpenSSL\bin\openssl.cnf** den entsprechenden Abschnitt an!

c. Erzeugen eines Zertifikats für Stefan Nagel und Signierung durch Marko Schuba.

Ergebnis: *nagel.crt*

```
Erzeugen des Schlüssels für Stefan Nagel
c:\OpenSSL\infoCA\user> openssl genrsa -des3 -out nagel_rsa.key
...

Erzeugen der Zertifikatsignierunganforderung für Stefan Nagel
c:\OpenSSL\infoCA\user> openssl req -new -key nagel_rsa.key -out nagel.csr
...

Signierung von Stefan Nagel's Anfrage durch Marko Schubas Schlüssel
C:\OpenSSL\infoCA\user>openssl x509 -req -in nagel.csr -extfile c:\openssl\bin\openssl.cnf -extensions
v3_ca  -CA schuba.crt -CAkey schuba_rsa.key -out nagel.crt -set_serial 7
...
```

Ok. 3 Zertifikate sind ausreichend um einige interessante Tests mit dem Befehl **verify** durchzuführen. Weitere Informationen zu den folgenden Ausführungen finden Sie im Anhang. Schauen Sie sich im Falle einer Fehlermeldung die entsprechende Nummer im Anhang an!

d. Verifizieren des kürzesten Zertifikatpfads, nur ein Zertifikat:

```
C:\OpenSSL\infoCA\user> openssl verify caadmin.crt
…
error 18 at 0 depth lookup:self signed certificate
OK

c:\OpenSSL\infoCA\user> openssl verify -CAfile caadmin.crt  caadmin.crt
caadmin.crt : OK

c:\OpenSSL\infoCA\user> openssl verify  schuba.crt
…
error 20 at 0 depth lookup:unable to get local issuer certificate

c:\OpenSSL\infoCA\user> openssl verify -CAfile  caadmin.crt  schuba.crt
…
schuba.crt: OK
```

Beachten Sie:

- Sie erhalten ein **OK** mit einem Fehler, wenn Sie ein selbstsigniertes Zertifikat prüfen, ohne es in die Liste der vertrauenswürdigen Zertifikate (CA Zertifikat) aufzunehmen.
- Sie erhalten ein perfektes **OK** bei der Prüfung eines selbstsignierten Zertifikats, wenn Sie es als CA Zertifikat (vertrauenswürdig) deklarieren. Die Vertrauenskette ist vom zu prüfenden Zertifikat, bis zum Root-CA Zertifikat durchgängig gegeben.
- Sie erhalten einen Fehler bei der Prüfung eines nicht selbstsigniertes Zertifikats, ohne ein CA Zertifikat zu deklarieren.
- Sie erhalten ein perfektes **OK** bei der Prüfung eines nicht selbstsignierten, sondern von der CA signierten Zertifikats, wenn Sie das CA Zertifikat deklarieren (Vertrauenskette komplett).

e. Verifizieren eines Zertifikatpfads mit zwei Zertifikaten:

```
C:\OpenSSL\bin> openssl verify -CAfile caadmin.crt nagel.crt
strauch.crt: /C=DE/ST=NRW/L=Aachen/O=FH Aachen/OU=ETIT/CN=Stefan Nagel
```

```
error 20 at 0 depth lookup:unable to get local issuer certificate

C:\OpenSSL\bin> openssl verify -CAfile schuba.crt nagel.crt
strauch.crt: /C=DE/ST=NRW/L=Aachen/O=FH Aachen/OU=ETIT/CN=Jakob Strauch
error 2 at 1 depth lookup:unable to get issuer certificate
```

Beachten Sie:

- Test 1: Pfad an der Stelle 0 (**nagel.crt**) unterbrochen. Kann den Herausgeber von **nagel.crt** (**schuba.crt**) nicht unter den vertrauenswürdigen Zertifikaten finden (Vertrauenskette unterbrochen).
- Test 2: Pfad an der Stelle 1 (**schuba.crt**) unterbrochen. Kann den Herausgeber von **schuba.crt** nicht finden.

f. Verifizieren eines Zertifikatpfads von vielen Zertifikaten:

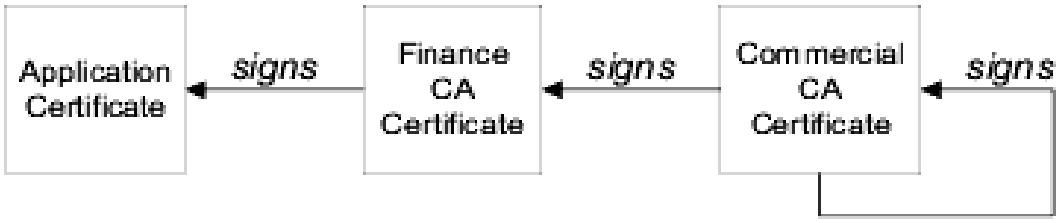
```
C:\OpenSSL\bin> openssl verify -CAfile caadmin.crt -untrusted schuba.crt nagel.crt
nagel.crt: OK
```

Beachten Sie:

- Test 1: Perfekt, die Vertrauenskette ist bis zur Root-CA gegeben (nagel->schuba->caadmin).

Zusammenfassung

Das Konzept des Zertifikatpfads ist einfach. Beachten Sie nur, dass ein Vorgängerzertifikat den Herausgeber des Folgezertifikats identifiziert.



Der OpenSSL-Befehl **verify** ist einfach zu verwenden, er benötigt nur zwei Parameter:
-CAfile und **-untrusted**.

Achtung, ab hier sollten Sie selbstständig die bisher gezeigten Befehle anwenden!

8. Erzeugung eines eigenen Zertifikats

a. Erzeugen Sie für sich selbst ein Zertifikat der CA *infoCA* mit einer Schlüssellänge von 1024 Bit. Verwenden Sie Ihre eigenen Daten (Ort, Name usw.) für das Zertifikat. Signieren Sie Ihr Zertifikat mit dem *infoCA* Schlüssel. Prüfen Sie anschließend den Zertifkatspfad zur Root-CA *infoCA*

Hinweis: Das Zertifikat der CA heißt *cacert.pem* (siehe Abschnitt 4 **Erzeugen des CA Root-Zertifikats** und liegt im Verzeichnis **c:\OpenSSL\infoCA** -> Sie müssen das *CAfile* als *..\cacert.pem* verwenden!

Führen Sie alle dazu benötigten Befehle hier auf:

Schlüsselerzeugung:
openssl genrsa -des3 -out test_rsa_des.key 1024

Zertifikatsignierungsanfrage:
openssl req -new -key test_rsa_des.key -out test.csr

Signierung:
openssl ca -in test.csr -out test.crt

Verifizieren:
C:\OpenSSL\infoCA\user>openssl verify -CAfile ..\cacert.pem test.crt
test.crt: OK

b. Schauen Sie sich die Komponenten Ihres Zertifikats an.

i. Welchen Befehl verwenden Sie dazu? **openssl x509 -in test.crt -noout -text**

ii. Welche Seriennummer hat Ihr Zertifikat? **2**

iii. Wer ist der Herausgeber Ihres Zertifikats? **CA FH-Aachen**

iv. Darf Ihr Zertifikat weitere Zertifikate signieren (woher nehmen Sie diese Information)?

X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE

c. Schauen Sie sich die CA-Dateien *index.txt* und *serial* an. Wie haben sie sich verändert?

index: 03

index.txt: neues Zertifikat zugefügt

d. Exportieren Sie Ihren öffentlichen Schlüssel. Geben Sie den benötigten Befehl an.

openssl rsa -in test_rsa_des.key -pubout -out test_rsa_des_pub.key

e. Exportieren Sie Ihr Zertifikat in das PKCS#12-Format. Geben Sie den benötigten Befehl an.

openssl pkcs12 -export -in test.crt -inkey test_rsa_des.key -out test.p12

Welches Export Passwort wollen Sie verwenden?

9. sTunnel

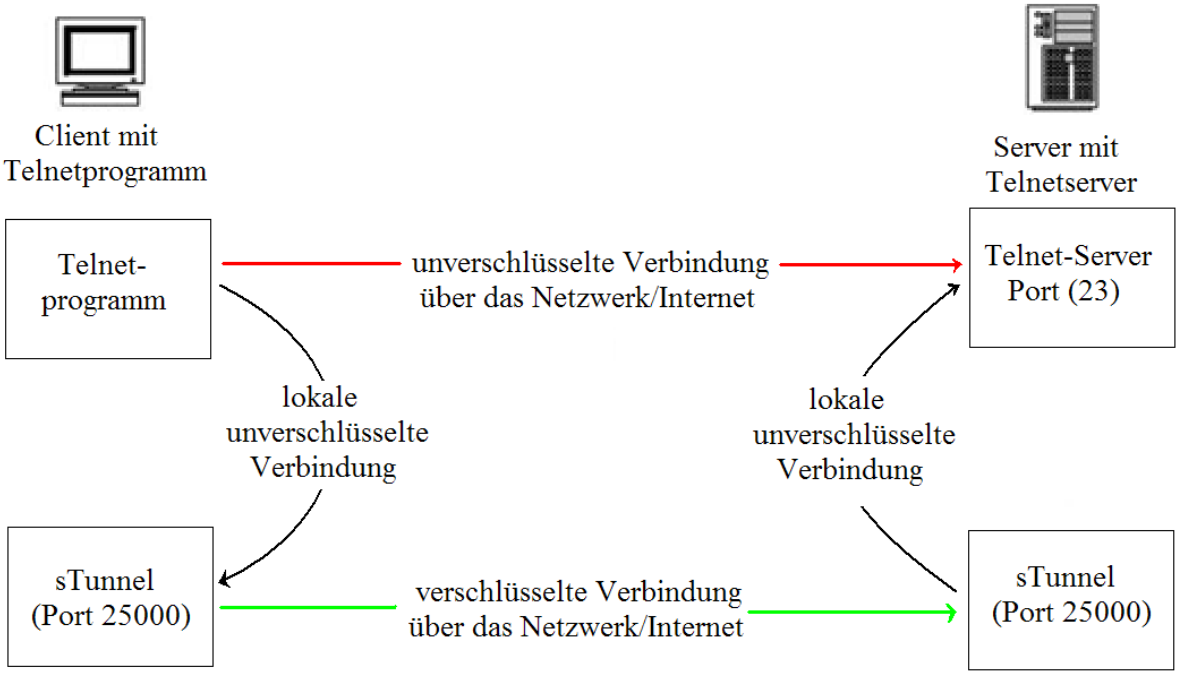
sTunnel ist ein einfaches Tool, das nur eine Sache kann, die aber sehr gut, es verschlüsselt eine Verbindung. Damit werden 2 Applikationen, die selber über keine Möglichkeit der Verschlüsselung verfügen, in die Lage versetzt, sicher (verschlüsselt) über das Netz zu kommunizieren. Mit sTunnel sind Sie in der Lage, innerhalb kürzester Zeit so eine Verbindung aufzubauen.

Starten Sie **VM-Server1**.

Starten Sie auf **VM-Client1** und **VM-Server1** die Datei **stunnel-4.21-installer.exe** (befindet sich entweder im Verzeichnis *c:\IS-Praktikum\stunnel* oder Sie können es von http://www.stunnel.org/download/binaries.html herunterladen). Installieren Sie in das vorgeschlagene Verzeichnis *c:\Programme\stunnel*.

Verschlüsseln von Telnet mit sTunnel

Nun sollen Sie Ihr selbst erstelltes Zertifikat auch anwenden. Dazu setzen wir das Programm **sTunnel** ein. Es soll eine Telnetsitzung (die normalerweise im Klartext übertragen wird, also für jeden Sniffer eine sichere Beute ist) mit SSL verschlüsseln. **sTunnel** wird auf dem Telnet-Client gestartet und lauscht am entsprechenden Port. Wenn Daten von einer Applikation auf dem Client ankommen, werden Sie zum Telnet-Server, an den zugehörigen Port (der eingestellte sTunnel-Port), weitergeleitet. Auf dem Server läuft eine weitere Instanz von **sTunnel**, der am Empfangsport lauscht und eingehende Daten an die lokale Telnetserverapplikation (Port 23) weiterleitet.



Erstellen Sie mit einem Texteditor, auf dem Client und auf dem Server, im Verzeichnis *c:\Programme\stunnel* die Datei infosec_cl.conf bzw. infosec_srv und kopieren Sie die unten angegebenen Zeilen in die Datei(en).

Konfigurationsdateidatei des Clients (infosec_cl.conf)

```
cert = c:/OpenSSL/infoCA/user/caadmin.crt
key = c:/OpenSSL/infoCA/user/caadmin_rsa.key
client = yes
ciphers = TLSv1:SSLv3:!SSLv2:!LOW:@STRENGTH

[telnet]
```

```
accept = 25000
connect = 10.0.0.3:25000
```

Sie müssen Ihr Client-Zertifikat und den zugehörigen Schlüssel angeben. Dann wird der Client-Modus eingestellt. Unter *ciphers* geben wir die unterstützten Verschlüsselungen an (TLSv1 und SSLv3 werden unterstützt, SSLv2 nicht). Der Parameter @STRENGTH gibt an, dass die Auswahl der verwendeten Verschlüsselung von STARK nach SCHWACH durchgeführt werden soll. Als lokaler Port wird Port 25000 ausgewählt (es kann jeder unbenutzte Port sein!). Dann müssen Sie den Telnetserver auf diesen Port umleiten. **Noch mal der Hinweis:** die Ports können frei gewählt werden, auch wenn hier im Beispiel jedes Mal Port 25000 verwendet wird!!

Konfigurationsdatei des Servers (infosec_srv.conf)

```
cert = /Programme/stunnel/schuba.crt
key = /Programme/stunnel/schuba_rsa_des.key
ciphers = TLSv1:SSLv3:!SSLv2:!LOW:@STRENGTH

[telnet]
accept = 25000
connect = 23
```

Diese Konfigurationsdatei sieht der Konfigurationsdatei des Client sehr ähnlich! Sie geben das Zertifikat und den Schlüssel des Servers an. Hier können Sie ein beliebiges Zertifikat, das schon vorher erstellt wurde, verwenden. Verwenden Sie Ihr eigenes Zertifikat, das Sie im letzten Schritt erzeugt haben. Im Beispiel verwende ich das Zertifikat und den Schlüssel von Marko Schuba. Die benötigten Dateien (schuba.crt und schuba_rsa.key) habe ich vorher vom Erzeugungsverzeichnis auf der anderen VM, auf dem diese Dateien liegen, in das lokale Verzeichnis c:\Programme\stunnel kopiert. Weiterhin legen sie die unterstützen Verschlüsselungen und die verwendeten Ports fest (eingehende Daten auf Port 25000 werden an den lokalen Port 23 (Telnetserver) weitergeleitet.

Nun können Sie sTunnel starten. Die Kommunikationspartner tauschen die Zertifikate aus (damit hat jeder den öffentlichen Schlüssel des Gegenübers) und ab geht die Post.

Öffnen Sie eine DOS-Box und wechseln Sie ins Installationsverzeichnis von sTunnel.
Geben Sie folgende Befehle ein:

auf dem Client: stunnel infosec_cl.conf
auf dem Server: stunnel infosec_srv.conf

Beim Start von sTunnel werden Sie nach dem Passwort der SSL-Schlüssel gefragt (Sie erinnern sich…die Dateien haben wir mit 3DES verschlüsselt). Geben Sie also die entsprechenden Pass Phrasen an!

Sniffen Sie die Telnetsession mit WireShark und sehen sich alles an und versuchen Sie die einzelnen Pakete zu interpretieren.

Dann starten Sie auf dem sTunnel-Client die Telnetapplikation: telnet localhost 25000

So, nun sollte alles verschlüsselt werden.

Anhang

DIAGNOSE von ‘verify’-Meldungen

Wenn eine ‘verify’-Operation fehlschlägt wird eine Fehlermeldung ausgegeben. Die allgemeine Form einer Fehlermeldung sieht folgendermaßen aus:

```
server.pem: /C=AU/ST=Queensland/O=CryptSoft Pty Ltd/CN=Test CA (1024 bit)
error 24 at 1 depth lookup:invalid CA certificate
```

Die erste Zeile enthält den Namen des Zertifikats, das geprüft wird. Anschließend wird der Zertifikatsbesitzer (subject) angezeigt. Die zweite Zeile enthält die Fehlernummer und den depth. Die depth gibt die Nummer des Zertifikats an, bei dessen Prüfung ein Problem auftrat. Es wird mit der Nummer 0 gestartet (0 ist das Zertifikat selbst, 1 die CA die es signiert hat usw.) Anschließend folgt der Fehlertext dieser Fehlernummer.

Eine ausführliche Liste der Fehlermeldungen folgt unten. Einige Fehlermeldungen wurden zwar definiert, aber nie verwendet: diese werden mit "unused" markiert. Es wurden nur die im Praktikum auftretenden Meldungen (0, 2, 18, 24) übersetzt

`0 X509_V_OK: ok`
: Die Operation war erfolgreich.

`2 X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT: unable to get issuer certificate`
: Das Herausgeberzertifikat kann nicht gefunden werden: dieser Fehler tritt auf, wenn das Herausgeberzertifikat eines nicht vertrauenswürdigen Zertifikats nicht gefunden werden kann.

`3 X509_V_ERR_UNABLE_TO_GET_CRL unable to get certificate CRL`
: the CRL of a certificate could not be found. Unused.

`4 X509_V_ERR_UNABLE_TO_DECRYPT_CERT_SIGNATURE: unable to decrypt certificate's signature`
: the certificate signature could not be decrypted. This means that the actual signature value could not be determined rather than it not matching the expected value, this is only meaningful for RSA keys.

`5 X509_V_ERR_UNABLE_TO_DECRYPT_CRL_SIGNATURE: unable to decrypt CRL's signature`
: the CRL signature could not be decrypted: this means that the actual signature value could not be determined rather than it not matching the expected value. Unused.

`6 X509_V_ERR_UNABLE_TO_DECODE_ISSUER_PUBLIC_KEY: unable to decode issuer public key`
: the public key in the certificate SubjectPublicKeyInfo could not be read.

`7 X509_V_ERR_CERT_SIGNATURE_FAILURE: certificate signature failure`
: the signature of the certificate is invalid.

`8 X509_V_ERR_CRL_SIGNATURE_FAILURE: CRL signature failure`
: the signature of the certificate is invalid. Unused.

`9 X509_V_ERR_CERT_NOT_YET_VALID: certificate is not yet valid`
: the certificate is not yet valid: the notBefore date is after the current time.

`10 X509_V_ERR_CERT_HAS_EXPIRED: certificate has expired`
: the certificate has expired: that is the notAfter date is before the current time.

`11 X509_V_ERR_CRL_NOT_YET_VALID: CRL is not yet valid`
: the CRL is not yet valid. Unused.

`12 X509_V_ERR_CRL_HAS_EXPIRED: CRL has expired`
: the CRL has expired. Unused.

`13 X509_V_ERR_ERROR_IN_CERT_NOT_BEFORE_FIELD: format error in certificate's notBefore field`
the certificate notBefore field contains an invalid time.

`14 X509_V_ERR_ERROR_IN_CERT_NOT_AFTER_FIELD: format error in certificate's notAfter field`
the certificate notAfter field contains an invalid time.

`15 X509_V_ERR_ERROR_IN_CRL_LAST_UPDATE_FIELD: format error in CRL's lastUpdate field`
the CRL lastUpdate field contains an invalid time. Unused.

`16 X509_V_ERR_ERROR_IN_CRL_NEXT_UPDATE_FIELD: format error in CRL's nextUpdate field`
the CRL nextUpdate field contains an invalid time. Unused.

`17 X509_V_ERR_OUT_OF_MEM: out of memory`
an error occurred trying to allocate memory. This should never happen.

`18 X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT: self signed certificate`
Das zu prüfende Zertifikat ist selbstsigniert und dasselbe Zertifikat kann nicht in der Liste der vertrauenswürdigen Zertifikate gefunden werden.

`19 X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN: self signed certificate in certificate chain`
the certificate chain could be built up using the untrusted certificates but the root could not be found locally.

`20 X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY: unable to get local issuer certificate`
the issuer certificate of a locally looked up certificate could not be found. This normally means the list of trusted certificates is not complete.

`21 X509_V_ERR_UNABLE_TO_VERIFY_LEAF_SIGNATURE: unable to verify the first certificate`
no signatures could be verified because the chain contains only one certificate and it is not self signed.

`22 X509_V_ERR_CERT_CHAIN_TOO_LONG: certificate chain too long`
the certificate chain length is greater than the supplied maximum depth. Unused.

`23 X509_V_ERR_CERT_REVOKED: certificate revoked`
the certificate has been revoked. Unused.

`24 X509_V_ERR_INVALID_CA: invalid CA certificate`
Ein CA Zertifikat ist ungültig. Entweder ist e keine CA oder die Erweiterungen stimmen nicht mit dem unterstützen Einsatzzweck überein.

`25 X509_V_ERR_PATH_LENGTH_EXCEEDED: path length constraint exceeded`
the basicConstraints pathlength parameter has been exceeded.

`26 X509_V_ERR_INVALID_PURPOSE: unsupported certificate purpose`
the supplied certificate cannot be used for the specified purpose.

`27 X509_V_ERR_CERT_UNTRUSTED: certificate not trusted`
the root CA is not marked as trusted for the specified purpose.

`28 X509_V_ERR_CERT_REJECTED: certificate rejected`
the root CA is marked to reject the specified purpose.

`29 X509_V_ERR_SUBJECT_ISSUER_MISMATCH: subject issuer mismatch`

the current candidate issuer certificate was rejected because its subject name did not match the issuer name of the current certificate. Only displayed when the **-issuer_checks** option is set.

30 X509_V_ERR_AKID_SKID_MISMATCH: authority and subject key identifier mismatch
the current candidate issuer certificate was rejected because its subject key identifier was present and did not match the authority key identifier current certificate. Only displayed when the **-issuer_checks** option is set.

31 X509_V_ERR_AKID_ISSUER_SERIAL_MISMATCH: authority and issuer serial number mismatch
the current candidate issuer certificate was rejected because its issuer name and serial number was present and did not match the authority key identifier of the current certificate. Only displayed when the **-issuer_checks** option is set.

32 X509_V_ERR_KEYUSAGE_NO_CERTSIGN:key usage does not include certificate signing
the current candidate issuer certificate was rejected because its keyUsage extension does not permit certificate signing.

50 X509_V_ERR_APPLICATION_VERIFICATION: application verification failure
an application specific error. Unused.