IT-Sicherheit Praktikum Nr.2 Wörterbuchangriffe und PKI mit OpenSSL

1. Passwort-Cracking mit Wörterbuchangriffen

Es sind immer noch einige unverschlüsselte Protokolle im Einsatz (z.B. FTP oder Telnet). Zum Ermitteln von Passwörtern, die mit diesen Protokollen übertragen werden, werden keine Wörterbuchangriffe benötigt. Ein Man-in-the-Middle Angriff und ein Sniffer sind ausreichend, um die Usernamen und Passwörter heraus zu bekommen. Erst bei verschlüsselten Protokollen sind Passwörter schwieriger (wenn überhaupt) findbar.
Wie ein sicheres Passwort aussieht sollten Sie mittlerweile wissen. Aber es ist schwierig, sich viele kryptische Passwörter zu merken. Daher werden, wider besseres Wissen, immer wieder relativeinfache Passwörter verwendet.

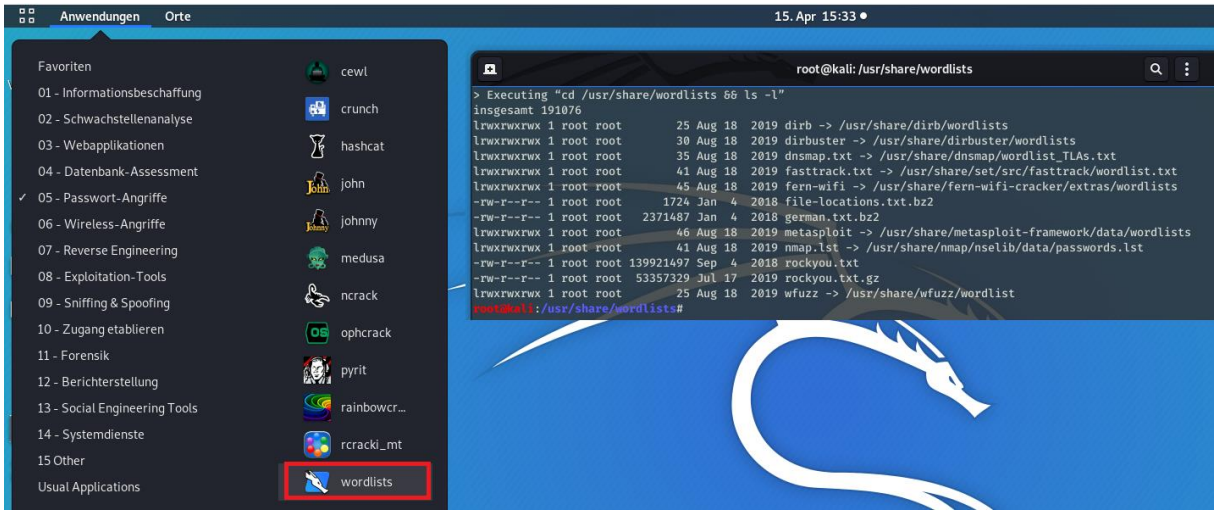
Um diese einfachen Passwörter zu kacken gibt es eine Menge Programme. Viele von diesen Programmen sind auf bestimmte Aufgaben hin optimiert. Je nachdem, ob Sie ein Passwort einer RAR-Datei, den Hash eines Windows-Passworts oder ein Login-Passwort auf einer Webseite suchen, gibt es besonders geeignete Programme.

Einige sollen hier im Praktikum einmal vorgestellt werden und Sie sollen sie einmal selber ausprobieren.

Die Pentester-Distribution **Kali** haben Sie schon kennen gelernt. Alle hier gezeigten Programme sind schon auf Kali installiert.

Sie werden im Weiteren Folgendes machen:
1. Herausfinden eines Windows-Passworts mit dem Programm **Hashcat**.
2. Knacken des Passworts einer verschlüsselten RAR-Datei mit dem Programm **John-the-Ripper**.
3. Angriff auf ein Webseiten-Login mit dem Programm **Hydra**.

Alle Angriffe werden als Wörterbuch-Angriffe durchgeführt. D.h. es werden vorgegebene Wörter getestet. Dabei verwenden wir hauptsächlich Wörterbücher, die schon auf Kali gespeichert sind. Nur das Wörterbuch *german.txt* kommt noch hinzu. Auf der FH Kali VM ist es schon gespeichert. Ansonsten können Sie es vom Ilias-Server herunterladen. Die Wörterbücher auf Kali liegen im Verzeichnis **/usr/share/wordlists**. Sie können sich die vorhandenen Wörterbücher über **Anwendungen->05-Passwort-Angriffe->wordlists** ansehen.



Ja, da sind schon eine Menge Wörterbücher drin. Die Wörterbücher, die komprimiert sind, müssen vor dem Gebrauch noch dekomprimiert werden (wie z.B. rockyou).

1. Herausfinden eines Windows-Passworts mit dem Programm *Hashcat*

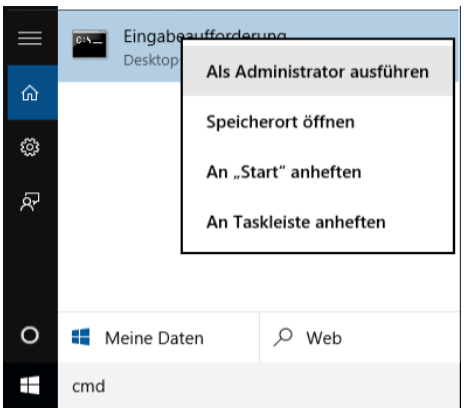
Starten Sie die VM **ITS2-Win10** und **Kali**. Prüfen Sie die IP-Adressen der VMs, sie sollten beide im 10.0.0.0/24 Netz liegen.

Userdaten:
ITS2-Win10: root/Passwort
Kali: root/toor

Die Passwörter der Windows-User liegen in einer Datei mit Namen **SAM**. Auf diese Datei darf nur ein Administrator zugreifen. Die Passwörter liegen aber nicht im Klartext vor, sondern werden vor dem Abspeichern gehasht.

Um ein Windows-Passwort zu knacken, muss zuerst der Passworthash aus dem SAM-File extrahiert werden. Dazu kann man das Programm **pwdump** verwenden. Für Windows 10 liegt es in der Version 7 vor (*pwdump7*). Das Programm **muss mit Admin-Rechten ausgeführt werden**. Dazu öffnen Sie ein Terminalfenster mit Adminrechten in Windows 10.

Geben Sie dazu links unten im Suchfenster *cmd* ein. Es wird das Programm Eingabeaufforderung angezeigt. Klicken Sie mit der rechten Maustaste darauf und wählen Sie **Als Administrator ausführen**.



Das (vorher heruntergeladene) Programm **pwdump7** liegt auf dem Desktop. Wechseln Sie in der Eingabeaufforderung in dieses Verzeichnis.

Starten Sie das Programm. Nach kurzer Zeit erhalten Sie Hashwerte aller auf dem System bekannten User.

```
C:\Users\root\Desktop>pwdump7
Pwdump v7.1 - raw password extractor
Author: Andres Tarasco Acuna
url: http://www.514.es

Administrator:500:NO PASSWORD*********************:31D6CFE0D16AE931B73C59D7E0C089C0:::
Gast:501:NO PASSWORD*********************:NO PASSWORD*********************:::
:503:NO PASSWORD*********************:NO PASSWORD*********************:::
root:1001:NO PASSWORD*********************:E19CCF75EE54E06B06A5907AF13CEF42:::
ITS2:1002:NO PASSWORD*********************:C312D1A33A561C8087D7F6F555CBD0C2:::
Selber:1003:NO PASSWORD*********************:3D11D5823D3D01ED20EC3FB40E0EB5C4:::
```

Kopieren Sie den Hashwert des gesuchten Benutzers (hier im Beispiel ITS2) auf die Kali VM (Copy&Paste) und speichern Sie es in der Datei **win.hash** ab.

Es soll das Wörterbuch **/usr/share/wordlists/german.txt** verwendet werden.

Verwenden Sie folgenden Befehl:

```
hashcat -m 1000 -a 0 -o /root/gefunden.txt --remove /root/win.hash
/usr/share/wordlists/german.txt --force
```

Das Programm startet…

Hinweis: schauen Sie sich mit *hashcat -h* die Bedeutung jedes Parameters an, es gibt eine Menge und wenn Sie die falschen Parameter verwenden werden Sie das Passwort nicht knacken können!

```
root@kali:~/Desktop# hashcat -m 1000 -a 0 -o /root/Desktop/gefunden.txt --remove win.hash /usr/share/wordlists/german.txt
hashcat (v5.1.0) starting...

* Device #1: Not a native Intel OpenCL runtime. Expect massive speed loss.
             You can use --force to override, but do not report related errors.
No devices found/left.

Started: Wed Apr 15 16:04:35 2020
Stopped: Wed Apr 15 16:04:35 2020
root@kali:~/Desktop# hashcat -m 1000 -a 0 -o /root/Desktop/gefunden.txt --remove win.hash /usr/share/wordlists/german.txt --force
hashcat (v5.1.0) starting...

OpenCL Platform #1: The pocl project
====================================
* Device #1: pthread-Intel(R) Core(TM) i9-8950HK CPU @ 2.90GHz, 1024/2955 MB allocatable, 3MCU

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

ATTENTION! Pure (unoptimized) OpenCL kernels selected.
This enables cracking passwords and salts > length 32 but for the price of drastically reduced performance.
If you want to switch to optimized OpenCL kernels, append -O to your commandline.

Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.

* Device #1: build_opts '-cl-std=CL1.2 -I OpenCL -I /usr/share/hashcat/OpenCL -D LOCAL_MEM_TYPE=2 -D VENDOR_ID=64 -D CUDA_ARCH=0 -D
AMD_ROCM=0 -D VECT_SIZE=8 -D DEVICE_TYPE=2 -D DGST_R0=0 -D DGST_R1=3 -D DGST_R2=2 -D DGST_R3=1 -D DGST_ELEM=4 -D KERN_TYPE=1000 -D _
unroll'
* Device #1: Kernel m01000_a0-pure.a9284b00.kernel not found in cache! Building may take a while...
Dictionary cache built:
* Filename..: /usr/share/wordlists/german.txt
* Passwords.: 688967
* Bytes.....: 8827974
* Keyspace..: 688967
* Runtime...: 0 secs


Session..........: hashcat
Status...........: Cracked
Hash.Type........: NTLM
Hash.Target......: c312d1a33a561c8087d7f6f555cbd0c2
Time.Started.....: Wed Apr 15 16:04:56 2020 (0 secs)
Time.Estimated...: Wed Apr 15 16:04:56 2020 (0 secs)
Guess.Base.......: File (/usr/share/wordlists/german.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  1354.1 kH/s (0.22ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 294912/688967 (42.80%)
Rejected.........: 0/294912 (0.00%)
Restore.Point....: 291840/688967 (42.36%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: Sesshaftigkeit -> Skorzewo

Started: Wed Apr 15 16:04:42 2020
Stopped: Wed Apr 15 16:04:57 2020
```

Das Resultat finden Sie in der Datei **gefunden.txt**.

```
root@kali:~/Desktop# cat gefunden.txt
c312d1a33a561c8087d7f6f555cbd0c2:Siegerpokal
```

Suchen Sie nun selber das Passwort des Benutzers **Selber** oben im Bild.

Wie lautet das Passwort des Benutzers Selber? **Zeltleinwand**

Mit dem Programm *hashcat* können natürlich auch andere Passworttypen gefunden werden, z.B. von Linux-Systemen. Bei Linux stehen die Passworte entweder in der Datei /etc/passwd oder in der Datei /etc/shadow. Hier sind die Hashwerte zusammengesetzt aus mehreren Informationen (z.B. auch der Salt-Wert).

Sie haben gesehen, wie schnell und einfach das Passwort gefunden wurde (wenn man ein passendes Wörterbuch hat!).

2. Knacken des Passwortes einer verschlüsselten RAR-Datei mit dem Programm *John-the-Ripper.*

Jetzt soll eine verschlüsselte RAR-Datei geknackt werden. Sie finden die Datei (datei1.rar) auf dem Ilias-Server im Verzeichnis für dieses Praktikum.
Bevor der Hash geknackt werden kann, muss er erst einmal aus der RAR-Datei extrahiert werden. Dazu wird das Programm **rar2john** verwendet (Hinweis: bei einer ZIP-Datei wird das Programm **zip2john** verwendet). Mit dem > Zeichen kann das Ergebnis direkt in eine Datei gespeichert werden (hier rar.hash).

```
root@kali:~/Desktop# rar2john Datei1.rar > rar.hash
```

```
rar2john Datei1.rar > rar.hash
```

Sie können sich den Hashwert mit *cat rar.hash* auch ansehen.

```
root@kali:~/Desktop# cat rar.hash
Datei1.rar:$rar5$16$e29fce112ba320d6d04c7990a67b8e36$15$993f77d941145873fb36e01ecf8cb11f$8$0b7d9a100e88825a
```

Das Knacken geschieht auf folgende Weise:

```
john rar.hash -wordlist=/usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
```

```
root@kali:~/Desktop# john rar.hash --wordlist=/usr/share/wordlists/dirbuster/directory-list-2.3-medium.tx
Using default input encoding: UTF-8
Loaded 1 password hash (RAR5 [PBKDF2-SHA256 256/256 AVX2 8x])
Cost 1 (iteration count) is 32768 for all loaded hashes
Will run 3 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
newsletter       (Datei1.rar)
1g 0:00:00:00 DONE (2020-04-15 16:34) 3.448g/s 662.0p/s 662.0c/s 662.0C/s 24..spam
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Wie Sie sehen wurde hier das Wörterbuch **/usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt** verwendet.

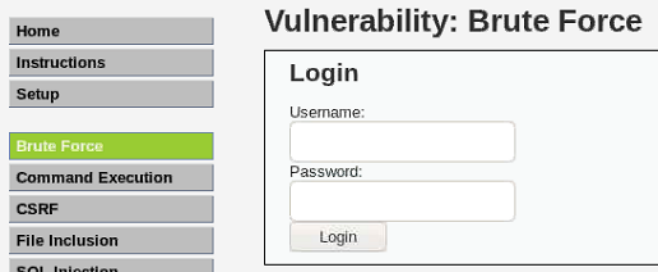
Laden Sie die Datei **Datei2.rar** vom Ilias-Server herunter und finden Sie das Passwort.

Wie lautet das Passwort? **TaskInfo2003_5**

3. Angriff auf ein Webseiten-Login mit dem Programm *Hydra*.

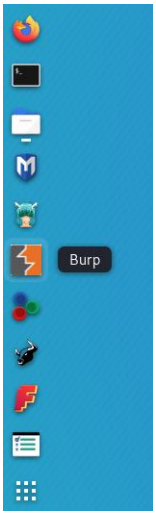
Bei dieser Aufgabe soll die schon aus dem Praktikum Nr.1 bekannte VM *Metasploitable* wieder zum Einsatz kommen. Starten Sie die VM und ermitteln Sie mit *nmap* die IP-Adresse.

Öffnen Sie auf *Kali* im Browser Firefox die Webseite *DVWA*. Loggen Sie sich dort mit **admin**/*password* ein. Stellen Sie den Sicherheitslevel von DVWA auf **low** und gehen Sie zu *Brute Force*.



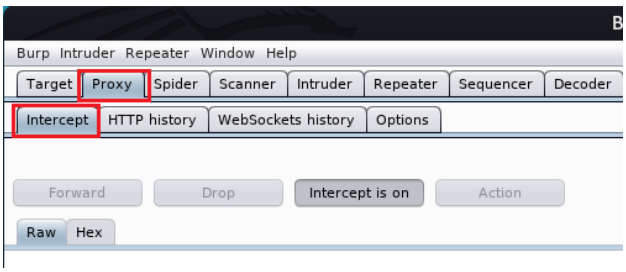
Sie sehen eine Login-Webseite. Dieses Login sollen Sie nun mit dem Programm *Hydra* mit einer Wörterbuch-Attacke knacken.

Das Programm Hydra benötigt einige Parameter, die leicht mit dem Proxy-Programm Burp zu ermitteln sind. Burp ist eine MitM-Applikation zwischen Browser und Ziel-Webserver. Starten Sie das Programm.

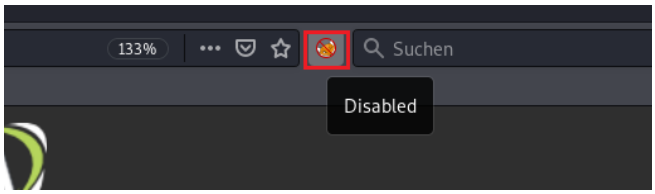


Hinweis: Führen Sie **KEIN** Update durch. Akzeptieren Sie ein temporäres Projekt mit **Next** und bleiben Sie dann bei den vorgegebenen Standardwerten.

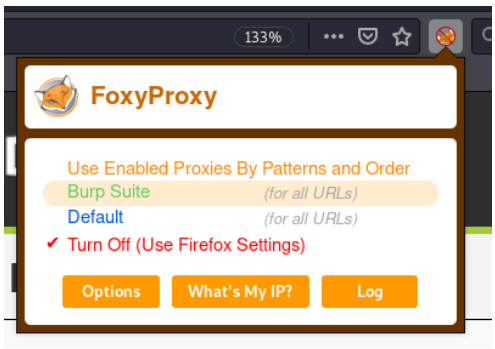
Gehen Sie dann auf den Reiter *Proxy/Intercept*. Es sollte auf **Intercept is on** stehen.

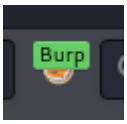


Nun müssen wir dem Browser Firefox noch mitteilen, dass er seine Daten alle an *Burp* sendet. Dafür ist auf der FH Kali VM schon das Programm FoxyProxy installiert und konfiguriert. Sie finden das ICON oben im Browser zwischen dem URL- und dem Suchenfenster.



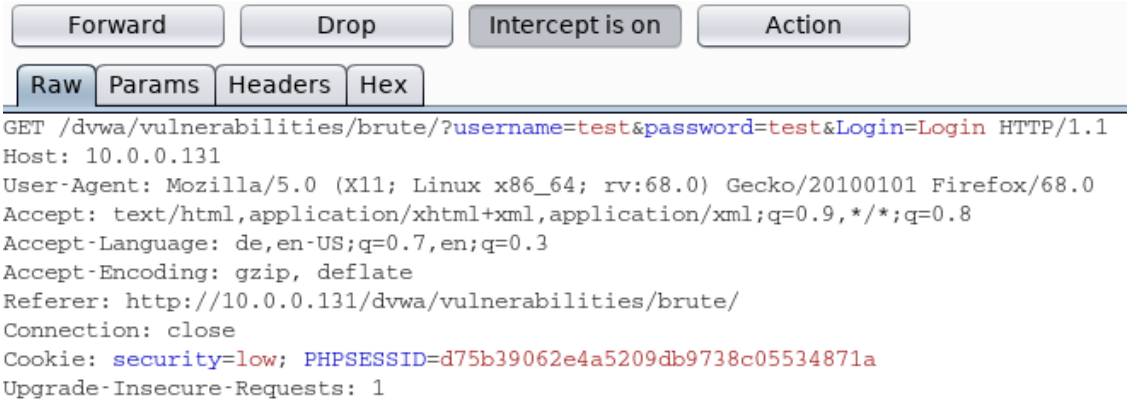
Klicken Sie auf das ICON und wählen Sie den grünen Eintrag **Burp Suite** aus.



Wenn die Auswahl übernommen wurde, ändert sich das ICON. 

Wenn Sie nun mit dem Browser Daten auf die Reise zum Webserver schicken, werden diese zuerst an das Programm Burp geleitet. Da hier Intercept is on ausgewählt ist, werden die Daten von Burp nicht sofort weitergeleitet. Sie können Sie sich ansehen, verändern und dann erst auf die Reise schicken. Geben Sie einmal einen beliebigen Benutzernamen und ein beliebiges Passwort ein und senden Sie es ab.

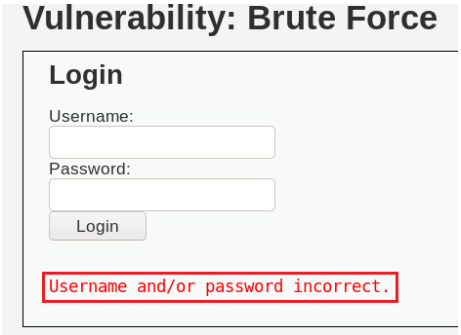
Sie erhalten keine Antwort, die Daten liegen noch bei Burp. Dort können Sie sie sich ansehen.



Sie können nun verschiedene für den Einsatz von Hydra wichtige Infos daraus ziehen:
1. Es handelt sich um einen http-**Get** Request.
2. Die genaue URL ist **/dvwa/vulnerabilities/brute**
2. Der übergebene GET Parameter lautet: **username=test&password=test&Login-login**
3. Es werden 2 Cookies mit übergeben: **security=low** und **PHPSESSID=….**

All diese Informationen werden als Parameter für Hydra benötigt. Aber noch etwas ist wichtig. Wenn die Passwörter ausprobiert werden, muss es ein Kriterium geben um feststellen zu können, ob der Versuch erfolgreich war oder nicht. Ein erfolgreicher Versuch würde bedeuten, dass wir schon ein gültiges Username/Passwort-Paar haben. Das ist meist nicht der Fall. Daher ist es besser ein Kriterium für den Mißerfolg zu haben. Das ist einfach. Wir geben einfach beliebige Dateien ein (z.B. wie hier im Beispiel ***test*/*test***). Wenn wir bei Burp die Unterbrechnugsfunktion ausschalten, wird die angezeigten Daten weitergeleitet und wir können auf der Webseite sehen, was passiert.

Klicken Sie auf *Intercept is on* und schauen Sie auf der Webseite nach.



Da haben wir die benötigten Informationen: **Username and/or password incorrect.**

Das wir kaum angezeigt werden, wenn der Loginversuch erfolgreich gewesen wäre.

So, nun haben wir alle Informationen für den Hydra-Befehl zusammen. Da wir allerdings weder einen Benutzernamen, noch ein Passwort haben, müssten wir auch beide suchen. Auch den Benutzernamen könnte man über ein Wörterbuch mit Namen suchen. Oft findet man auf der Webseite auch Name (ob in Email-Adressen oder sonst wo), die man zuerst versuchen kann. Das wollen wir auch hier im Praktikum machen. Da alle Benutzernamen mit allen Passwörtern kombiniert werden müssten, dauert das hier im Praktikum zu lange.

Wir verwenden als Usernamen **admin**.

Der Befehl lautet (IP Adresse von DVWA im Beispiel ist 10.0.0.131):

```
hydra 10.0.0.131 -l admin -P /usr/share/wordlists/fasttrack.txt http-get-form
"/dvwa/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:Username and/or
password incorrect.:H-Cookie: security=low;PHPSESSID=d75b39062e4a5209db9738c05534871a" -V
```

Bedeutung der Parameter können Sie sich mit *-h* ansehen. Beim Cookie **PHPSESSID** können Sie den Wert einer neuen Anfrage, die von Burp abgefangen wurde (*Achtung wieder auf Intercept is ON umstellen*), eintragen (Copy&Paste).

Welches Passwort erhalten Sie für *admin*? **password**

Es gibt noch einen weiteren Benutzer, der das gleiche Passwort verwendet. Auf Ilias finden Sie eine (kurze) Namensliste (namen.txt).

Welcher Benutzer verwendet noch das Passwort von *admin*? **smithy**

Befehl: hydra 10.0.0.131 -L namen.txt -p password http-get-form "/dvwa/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H-Cookie: security=low;PHPSESSID=d75b39062e4a5209db9738c05534871a" -V

2. PKI mit Open SSL

In diesem Teil 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.

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

a. Installation von OpenSSL

Starten Sie die VirtualBox. Starten Sie die VM **Client1**.

Hinweis: Sollte die VM nicht vorhanden sein, müssen Sie sie hinzufügen. Dazu klicken Sie auf *Maschine->hinzufügen*, gehen Sie in das Verzeichnis *D:\VMs\ITS\Client1* und wählen Sie die Datei **Client1**.

Loggen Sie sich ein (infosec/P@ssw0rd).

Starten Sie das Programm **Win32OpenSSL-1_0_1h.exe** im Verzeichnis *c:\ITS-Praktikum\OpenSSL*.

Geben Sie als Zielverzeichnis der Installation *c:\OpenSSL* an. Verwenden Sie ansonsten die vorgegebenen Standarteinstellungen.

Entfernen Sie Im Fenster *Completing the OpenSSL (32-bit) Setup Wizard* den Haken bei der ersten Auswahlmöglichkeit und klicken Sie dann auf **Finish**.

Öffnen Sie eine DOS-Box (als Administrator) und wechseln Sie im Installationsverzeichnis ins Unterverzeichnis *bin*. Geben Sie folgenden Befehl ein:

```
C:\OpenSSL\bin>openssl -help
WARNING: can't open config file: /usr/local/ssl//openssl.cfg
openssl:Error: '-help' is an invalid command.

Standard commands
asn1parse       ca              ciphers         cms
crl             crl2pkcs7       dgst            dh
dhparam         dsa             dsaparam        ec
ecparam         enc             engine          errstr
```

```
gendh           gendsa          genpkey         genrsa
nseq            ocsp            passwd          pkcs12
pkcs7           pkcs8           pkey            pkeyparam
pkeyutl         prime           rand            req
rsa             rsautl          s_client        s_server
s_time          sess_id         smime           speed
spkac           srp             ts              verify
version         x509

Message Digest commands (see the `dgst' command for more detail
md4             md5             mdc2            rmd160
sha             sha1

Cipher commands (see the `enc' command for more details)
aes-128-cbc     aes-128-ecb     aes-192-cbc     aes-192-e
aes-256-cbc     aes-256-ecb     base64          bf
bf-cbc          bf-cfb          bf-ecb          bf-ofb
camellia-128-cbc camellia-128-ecb camellia-192-cbc  camellia-
camellia-256-cbc camellia-256-ecb 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-o
des-ede3        des-ede3-cbc    des-ede3-cfb    des-ede3-
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          seed            seed-cbc        seed-cfb
```

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

Erstellen Sie zunächst einige Verzeichnisse unter *c:\OpenSSL*. Diese Verzeichnisse werden für den Betrieb einer CA benötigt.

- *itsCA* (Rootverzeichnis für unsere CA)
- *itsCA\certs* (Hier kommen die Zertifikate rein)
- *itsCA\private* (Hier landet der private Schlüssel der CA)
- *itsCA\newcerts* (Hier kommen neu erstellte Zertifikate rein)
- *itsCA\user* (Hierher kommen Zertifikatsignierungsanfragen, Benutzerzertifikate usw.)

Erstellen Sie im Verzeichnis *itsCA* die Textdatei **serial** (ohne Extension!) und schreiben Sie **01** hinein. Das ist die Seriennummer des nächsten Zertifikats, das von der CA erstellt wird.

Erstellen Sie dann im gleichen Verzeichnis (*itsCA*) die leere Textdatei **index.txt**

In den oben erzeugten Verzeichnissen werden Zertifikate und Schlüssel gespeichert. Der Basispfad (hier *itsCA*) kann einen beliebigen Namen haben, die Unterverzeichnisse sollten so heißen, wie hier dargestellt (dann passt das mit der Voreinstellung in der Konfigurationsdatei). Sie müssen nur nachher darauf achten, den richtigen Basispfad in die Konfigurationsdatei (*openssl.cfg*) einzutragen. 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 *itsCA\user* dient als Verzeichnis für die Benutzerzertifikate.

b. Erzeugen der CA *itsCA*

Der nächste Schritt ist eine Anpassung der OpenSSL-Konfigurationsdatei openssl.cfg im Verzeichnis **c:\OpenSSL\bin** notwendig**.** Öffnen Sie diese Datei mit ***Wordpad***.

Die Datei **openssl.cfg** ist in mehrere Sektionen aufgeteilt. Dadurch kann man mehrere Zertifikatsstellen auf einem Rechner verwalten. Im Beispiel benennen wir einfach die Default-CA Sektion um und verwenden sie für unsere CA *itsCA*.

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 = itsCA # The default ca section
##################################################################

...

[ itsCA ]
dir = c:/OpenSSL/itsCA # 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 dem Eintrag **dir** auf die Vorwärts-Slashes!

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

<u>**Starten Sie die VM nun neu (nicht zurücksetzen!) und loggen Sie sich wieder ein:**</u> (infosec/P@ssw0rd).

c. 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 Algorithmus: Ron **R**ivest, Adi **S**hamir und Leonard **A**dleman. Heute ist RSA der im Internet am 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\itsCA\user* gespeichert werden. Öffnen Sie eine DOS-Box und wechseln Sie ins Verzeichnis *c:\OpenSSL\itsCA\user*.

Da keine Schlüssellänge angegeben ist, wird die Standardlänge von 1024 verwendet (in *openssl.cgf* angegeben).

Sie erzeugen zuerst einen Schlüssel, dann sehen Sie ihn sich an.

```
c:\OpenSSL\itsCA\user> openssl genrsa -out hoefken_rsa.key
Loading 'screen' into randomstate - done
Generating RSA private key, 1024 bit long modulus
…………………………………………………………...++++++++++++
................++++++++++++
e is 65537 (0x10001)

c:\OpenSSL\itsCA\user> type hoefken_rsa.key
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQC/kx0yV2tJsnknZv9E+Dpg5993F9d5aecmGRJ6rh2arI/fGH0T
3c6yl1D7WuqP9VPOvzXWJqDuTy1HylITwIpj8jy7Gd9WHvSYF6d+fDGqx4xNWKf8
2HXAKNV25U0UtplYtPK0nypDwknPwyt9o2UF59LZjzdYTudMKbpvS4UvyQIDAQAB
AoGAMxpBLY6YNf5/xvz4lyNIH1V7DL+1YNPZLxZ9EXCkwP/LGQ3lP4lpe8ZVeddU
vrYIdxwWdluzjh5cy8uyfUZdI7m1r2E/qPkTWqt25m8PKhFiw9wlcUjsEeSKy+Hh
Hfy/ZVN2Nzdtg9l9WbVZpxLw5MaPgP2p0wJu6wF9tWSU3skCQQDr7LnP42eRJjKV
LCr8zUGq9epAVXEDe7pANenZmDZDFTrRODoXvtiMjziSAbxzSK7L8s8utNGe9x2U
XDUqXm1XAkEAz+BJleknzL1iZKxxQQunVsxf46yVZ04nSYZUEnUNz4uzefMUu05R
XSHICG8SRpMTMyAKLyYdbNAXYchhZkP33wJAW3GkC9n+uJnvnTZDaz07qMIsi8cS
SOFaLa03+OkwjwvZgmKi0bVdRg31RVJ68sxHRDHW8CmR3mW2B10+FjAHfQJBAMUz
5RsOIO+eCiWvYZFvWMSDBcSbCPEDt0mM4VzRJtTpls3tyJEYbx+G2JkHpbLF+8TY
dFDeO7PqhQQCcEPv3skCQHt2TfaiaF3OlXduo1ztA27vhNpqS8HO3M4stC5LxRFX
sHRh+WxNvxGgop/YIEPIM+47BAk7c5moBxjTCUZTdfI=
-----END RSA PRIVATE KEY-----
```

Hinweis:

- Die Ausgabe zeigt nur einen privaten Schlüssel an, aber es wird sowohl ein privater, als auch ein öffentlicher Schlüssel erzeugt, also ein Schlüsselpaar!
- Das Schlüsselpaar wird im Format **PEM** erzeugt (auch wenn die Dateierweiterung **.key** lautet, das war unsere eigene Entscheidung!).
- Die Datei liegt im Verzeichnis c:\OpenSSL\itsCA\user

Anzeigen der RSA Schlüsselkomponenten

So können Sie alle Schlüsselkomponenten anzeigen:

```
c:\OpenSSL\itsCA\user> openssl rsa -in hoefken_rsa.key -text
Private-Key: (1024 bit)
modulus:
    00:db:f5:3f:35:da:f7:ef:3f:b8:71:e3:10:2e:a0:
    51:fe:ce:99:5f:2e:6b:2a:f9:f7:38:c9:13:99:86:
    96:59:f2:df:51:58:aa:c5:c4:3c:f5:bc:7e:85:89:
    c2:93:c8:fa:d5:ef:97:05:e8:b4:dc:ec:3b:95:45:
    16:a9:b8:25:c2:44:35:94:2f:17:6e:c5:e6:bf:4d:
    c6:71:4d:a8:3c:e0:30:41:7c:e2:dc:c1:84:78:81:
    e4:d7:c4:b8:2c:86:e6:f9:1f:32:46:52:e0:a3:f7:
    85:b1:e6:cd:ae:8e:5e:93:2b:1b:60:68:30:5d:5f:
    8b:2a:c2:88:9b:20:d4:70:b3
publicExponent: 65537 (0x10001)
privateExponent:
    79:47:1c:e6:2d:e8:bc:a2:ce:a9:04:c0:7c:64:eb:
    30:ea:6b:08:5f:08:b9:f6:7f:48:71:1d:fc:6d:87:
    e2:ea:96:15:3d:25:53:cb:e2:ac:bf:94:a7:3e:90:
    e9:79:4c:4c:bf:40:ef:02:23:0f:18:b5:b5:95:15:
    6c:d8:da:ff:df:3b:7c:03:37:b8:42:f3:ba:dc:dd:
    16:3d:7c:85:70:b1:b9:58:34:32:11:43:cb:e0:20:
    d7:c7:3c:28:42:b5:d7:e5:34:84:5d:35:b8:86:99:
    f0:fe:8a:b5:e1:b5:69:d3:0f:86:0d:40:1a:06:38:
    88:8a:16:aa:57:59:6f:a9
prime1:
    00:f7:f8:4e:27:37:84:28:99:ed:fd:91:7b:32:8f:
    58:a8:61:8a:37:03:19:09:8a:70:c0:24:85:60:10:
    86:9e:4b:fd:91:c3:ca:bc:6f:ca:e4:83:43:ee:26:
    96:93:a0:d5:3c:bc:dc:64:f4:62:63:07:4d:ca:29:
    7a:cc:db:d7:e5
prime2:
    00:e3:14:b8:4a:da:95:e2:b8:b9:3f:ff:6f:d7:b4:
    22:b7:60:b1:52:8b:64:8c:78:b1:a1:6d:2d:a5:25:
    5c:1c:1f:9b:ab:7e:d6:41:f1:5e:db:43:d6:42:6b:
    48:6f:4d:14:68:a7:2e:59:11:cd:de:a6:58:68:2a:
    0b:bb:ae:ec:b7
exponent1:
    3c:72:a3:5a:9c:05:f3:7f:d2:ac:45:92:aa:b8:96:
    fc:35:0a:e4:6e:34:e4:46:83:45:d3:a1:4f:d1:b7:
    7f:4f:36:f9:19:a9:8a:56:16:37:ae:68:18:dd:ff:
    46:ed:a6:0d:b0:5c:69:9f:c9:7c:2a:83:a4:0c:88:
    8a:f7:ec:a1
```

```
exponent2:
    00:9e:d5:0f:52:cc:31:aa:b3:aa:5f:a9:3b:81:94:
    ed:f3:76:1e:91:ae:0e:56:72:d5:4c:24:21:99:31:
    d8:8b:93:50:3f:54:28:69:57:7b:11:c3:11:4a:c3:
    df:96:7a:43:9c:2f:d3:ab:59:58:b6:33:84:02:d7:
    e5:fd:11:0d:61
coefficient:
    05:4c:4a:95:a3:96:f6:ff:05:d0:ea:90:61:92:13:
    f8:f0:f0:81:9f:16:e6:07:34:c9:b4:34:51:99:58:
    15:95:89:0f:a0:4b:25:3b:98:fb:95:79:33:49:82:
    e9:df:7a:da:31:59:6c:0e:eb:b8:bb:64:ec:b2:a4:
    e5:7e:81:71
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDb9T812vfvP7hx4xAuoFH+zplfLmsq+fc4yROZhpZZ8t9RWKrF
xDz1vH6FicKTyPrV75cF6LTc7DuVRRapuCXCRDWULxduxea/TcZxTag84DBBfOLc
wYR4geTXxLgshub5HzJGUuCj94Wx5s2ujl6TKxtgaDBdX4sqwoibINRwswIDAQAB
AoGAeUcc5i3ovKLOqQTAfGTrMOprCF8IufZ/SHEd/G2H4uqWFT0lU8virL+Upz6Q
6XlMTL9A7wIjDxi1tZUVbNja/987fAM3uELzutzdFj18hXCxuVg0MhFDy+Ag18c8
KEK11+U0hF01uIaZ8P6KteG1adMPhg1AGgY4iIoWqldZb6kCQQD3+E4nN4Qome39
kXsyj1ioYYo3AxkJinDAJIVgEIaeS/2Rw8q8b8rkg0PuJpaToNU8vNxk9GJjB03K
KXrM29flAkEA4xS4StqV4ri5P/9v17Qit2CxUotkjHixoW0tpSVcHB+bq37WQfFe
20PWQmtIb00UaKcuWRHN3qZYaCoLu67stwJAPHKjWpwF83/SrEWSqriW/DUK5G40
5EaDRdOhT9G3f082+RmpilYWN65oGN3/Ru2mDbBcaZ/JfCqDpAyIivfsoQJBAJ7V
D1LMMaqzql+pO4GU7fN2HpGuDlZy1UwkIZkx2IuTUD9UKGlXexHDEUrD35Z6Q5wv
06tZWLYzhALX5f0RDWECQAVMSpWjlvb/BdDqkGGSE/jw8IGfFuYHNMm0NFGZWBWV
iQ+gSyU7mPuVeTNJgunfetoxWWwO67i7ZOyypOV+gXE=
-----END RSA PRIVATE KEY-----
```

Hinweis:
In dieser Datei befinden sich **2** Schlüssel, Ihr privater und Ihr öffentlicher Schlüssel. Sie können die einzelnen Komponenten (Primzahl, Koeffizient usw.) identifizieren. Am Ende sehen Sie wieder ihren privaten Schlüssel.

Verschlüsseln der RSA Keys

Die RSA Schlüssel beinhalten auch den *Private Key*, der niemand anderem als Ihnen bekannt sein sollte. Daher wird empfohlen, die RSA Schlüssel in verschlüsselter Form (Passwort geschützt) abzuspeichern (aktuell ist die Datei unverschlüsselt!).

In Folgenden wird gezeigt, wie Sie eine RSA Schlüsseldatei, verschlüsselt mit dem AES Algorithmus, erzeugen. Hier wird eine Schlüssellänge von 2048 Bit gewählt. Verwenden Sie als Passwort (Pass Phrase) hier im Praktikum die Zusammensetzung ***name****key* also in folgenden Beispiel: Name=hoefken -> Pass Phrase: *hoefkenkey*. Verwenden Sie ihren eigenen Namen!

Zeigen Sie den erzeugten Schlüssel wieder an, dieses Mal allerdings ist er nicht im Klartext!

```
c:\OpenSSL\itsCA\user> openssl genrsa –aes192 -out hoefken_rsa_aes.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_aes.key: hoefkenkey
```

```
Verifying - Enter pass phrase for hoefken_rsa_aes.key: hoefkenkey


c:\OpenSSL\itsCA\user> type hoefken_rsa_aes.key

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-192-CBC,CFFD309C4F5AA4F86C2E7E87B04CB58E

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-----
```

Hinweis: Es wurde eine 2. Datei (*hoefken_rsa_aes.key*) im Verzeichnis C:\OpenSSL\itsCA\user erzeugt. Schauen Sie einmal nach.

Perfekt! Der Extraparameter **–aes192** veranlasst den **genrsa** Befehl, das Schlüsselpaar zu erzeugen und die erzeugte Datei direkt mit dem AES-192-CBC Algorithmus zu verschlüsseln.

Dieser Befehl kann bei der Erzeugung des Schlüsselpaares verwendet werden. Falls die Schlüsseldatei schon existiert, können Sie auch den **rsa** Befehl verwenden um sie zu verschlüsseln.

Das probieren Sie nun einmal aus. Sie verschlüsseln die Datei *hoefken_rsa.key* (die wir zuerst erzeugt haben, und die unverschlüsselt ist*)*, die mit dem 1024 Bit langen RSA-Schlüssel erzeugt wurde. Sie überschreiben die im letzten Schritt erzeugte Datei (hoefken_rsa_aes.key). Da das Ergebnis wieder verschlüsselt wird, müssen Sie wieder eine Pass Phrase eingeben (s.o.)!

```
c:\OpenSSL\itsCA\user> openssl rsa -in hoefken_rsa.key –aes192 -out hoefken_rsa_aes.key
writing RSA key
Enter PEM pass phrase: hoefkenkey
```

```
Verifying - Enter PEM pass phrase: hoefkenkey



c:\OpenSSL\itsCA\user> type hoefken_rsa_aes.key

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-192-CBC,AFA17B778ED45D97E530D4B541ACDD21

gUlxTKy6MTIaqzNJRWJ3qYdfKBpRjsZPpvyCprsWjD1sbVK8vI6EKfz9uxX0VD0u
SVRCoezU9csgpCHmQ5gYG4U+HyMOzvzhez+UcBV9uP3gzmG5C5OmHTNSTz+pV54t
Y+aBLJWWO+U2rWbHShYBsG94XGbFfGAj3nawnvvk4Eh2CLT4aMGo/xzuM75TKwZz
+GrOYV9ydLYbG3Ed1WsLY+LQpif6f+G5UHdLWTDwXEOxRbRQDSjw1Jy6JwDe24Te
jDhBaVWutvetfc6YaYSXMRn/Fz1Jkz0g3cga7SxfgIRcn2DAGHwO1veWI1Jb38+j
HQQv5n+mYCQWamK7sob2nnoomL+yxoNj1D0rD9ltbkewD9jYaiDm+dWVCA1KygOr
JEKBzTT/CaB8eHzJPqmNoSPP8YpKDeA0pEuIDDqPFWQM5Un+C+Heew2sms6Jb++d
NHStV/8qG4LbDo4KOI2RxNU3/XsVSKjO/iwIKCGAH3flgpI06BSn6jj9gtjewEMS
+o80UIqlU8hp2dYGEnQ5JIeXhqBLZctb+Uf6ovHmdxzbX4jno3LJ1fziqdEw3aQF
QZgo8mp1pnJzGuvftrge66S8HWTaftXw3NFHJuWHT8ALfmFb0ME7vYh7UI6BguOU
An7CCVTsGJ+PvippQ2hmNWTi1fAG6NdFTqSESnDgflsM31YDt4tMz4+0h/WrA+M2
l/FG1feI0jYXvwTkrQ+plQ8+EREY8OsHjGnqjP56u1qiRjjvmW7FqC080VPzp1yT
4yDrSHsizn7Uy3MI8yNLdFRgM7GiB9Dxh+z3zBcp4AwePaVkX6D0vWpM34fOjxV2
-----END RSA PRIVATE KEY-----
```

Wie sie sehen, ist die Datei nun auch verschlüsselt! (DEK-Info: …)

Als nächstes werden wir aus der eben erzeugten Datei den Public Key extrahieren. ***Wenn ein Schlüsselpaar auch zur Authentifizierung genutzt werden soll, sollten Private Key und Public Key in getrennten Dateien liegen!*** Da die Datei verschlüsselt ist, müssen wir bei dieser Aktion unser Passwort eingeben.

```
c:\OpenSSL\itsCA\user> openssl rsa -in hoefken_rsa_aes.key -pubout -out hoefken_rsa_aes_pub.key
Enter pass phrase for hoefken_rsa_aes.key: hoefkenkey
Writing RSA key

c:\OpenSSL\itsCA\user> type hoefken_rsa_aes_pub.key

-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC/kx0yV2tJsnknZv9E+Dpg5993
F9d5aecmGRJ6rh2arI/fGH0T3c6yl1D7WuqP9VPOvzXWJqDuTy1HylITwIpj8jy7
Gd9WHvSYF6d+fDGqx4xNWKf82HXAKNV25U0UtplYtPK0nypDwknPwyt9o2UF59LZ
jzdYTudMKbpvS4UvyQIDAQAB
-----END PUBLIC KEY-----
```

Hinweis: Es wurde nun eine neue Datei (*hoefken_rsa_aes_pub.key*) erstellt, in der sich nur der öffentliche Schlüssel aus der Datei *hoefken_rsa_aes.key* befindet. Diese Datei ist unverschlüsselt!

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

Zusammenfassung

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

d. 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 **immer** signiert ist. Mit diesem Zertifikat sind Sie dann in der Lage, andere Zertifikate auszustellen und zu verifizieren.

Verwenden Sie als Pass-Phrase: **INFOSEC**

Geben Sie unten die entsprechenden Daten ein (einige haben wir schon in die **openssl.cfg** 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: ETIT
Common Name: CA
Email Address: ca@fh-aachen.de

Geben Sie folgenden Befehl ein:

```
c:\OpenSSL\itsCA\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: INFOSEC
Verifying - Enter PEM pass phrase: INFOSEC
-----
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]: (return)
Common Name (e.g, server FQDN or YOUR name) []:CA
Email Address []:ca@fh-aachen.de


c:\OpenSSL\itsCA\user> type ..\private\cakey.pem
-----BEGIN RSA PRIVATE KEY-----
MIICxjBABgkqhkiG9w0BBQ0wMzAbBgkqhkiG9w0BBQwwDgQI+HB5Nknr+0wCAggA
MBQGCCqGSIb3DQMHBAjjHDlU0UNXhASCAoBZENRqsAGYz1EzFwCM6vGrUoRw1Crl
IiNNiCYRhITJ4LlNr3xKinPJL7JfunEeGbeBZxVRWd8jAHxi1nAtMsCGSBgaHXhy
OdkEJPl+8loFJLr6huFAJSFosbyh5iGzpH9s3TZDsrGqF8mkE1UHkCoxQWmVN9ES
MZ1c5rKE/E6dLq6zChBsuervDg1PfJOTnhiPnB1+Vmc6i/64VxARmV2zpl9M1bDg
9xiyh9YcacgBmBswtWTFFXc09BkqbacM8SkbD8DdAsxljlfombqIINk/gQHI5Djh
0mQuxhHwmFWS1Um3zr5jJ/CNvg7tjbRndUKzJBjTmSIPe/J1O7mMBAFs4Joci3yV
```

```
lSxh63ZpTfF/YoLFGFZ+xOD7rT4JCQtCi8qdePW3SFJEJY3/OeLXWsnzL8nexk+I
pF4ZPPVPYlkJHKXvijh0VvpObL96ihuwAfFATZhzNB4ocasEn+le7z3sJvWFQozj
fEAXs19T+tOw6POWshBFsPZ0yPI2+vBNJQcEOHuwNBfyefwC2mhz5ScJIC3dsSmJ
d6zHdqHCgqrwvHyPxCQbbykgF264I82ohDoSwGIxQtvce2DqZnY3vnmHUbsu95OC
zNVT+s1d9QmpxTjZoN+ztCShZZzAlmsf8h+Mr1SNIoNpLaSBsAikuQmOu42yLkLa
qw8xgqL5WIdPooPFlS+bGoIhoNcMzkFVxwNP+S7cH9o9RV7dNR5LhijZxeJRxGc8
Yy80vIzI1aWXr4IPdeYEuvLFcdFZBUsvxhitXC5zR+oh8Qf66RHn0/jRN6K+iKCb
p2Oo8NEn3cIAe1kHOa41sXDWUfJRmFmxBPsPubqk74pFWhOFkJCaLpPO
-----END RSA PRIVATE KEY-----

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

Bis wann ist das Zertifikat gültig (genaues Datum)? *unterschiedlich*

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.

e. 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* 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 (i.d.R. erstellt mit dem privaten Schlüssel des Herausgebers).
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 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 Schlüsselpaar-Datei **hoefken_rsa_aes.key**, die wir früher schon erstellt haben, zu erzeugen.

Hinweis: Da die Datei verschlüsselt ist, müssen Sie wieder das Passwort vor der Verwendung eingeben.

```
c:\OpenSSL\itsCA\user> openssl req -new -key hoefken_rsa_aes.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]: (return)
Common Name (eg, YOUR name) []:Hans Hoefken
Email Address []:hoefken@fh-aachen.de

c:\OpenSSL\itsCA\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:

- Es wurde ein digitales Zertifikat erstellt und in der Datei *hoefken.crt* gespeichert.
- Der öffentliche Schlüssel, der in das Zertifikat aufgenommen wird, wird der RSA Schlüsselpaar-Datei **hoefken_rsa_aes.key** entnommen.
- Der private Schlüssel, mit dem das Zertifikat signiert wird, wird ebenfalls der RSA Schlüsselpaar-Datei **hoefken_rsa_aes.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\itsCA\user> openssl x509 -in hoefken.crt -noout -text
Certificate:
   Data:
```

```
    Version: 3 (0x2)
    Serial Number:
        d1:81:8a:e4:89:b3:63:3e
  Signature Algorithm: sha1WithRSAEncryption
    Issuer: C=DE, ST=NRW, L=Aachen, O=FH Aachen, OU=ETIT, CN=Hans
            Hoefken/emailAddress=hoefken@fh-aachen.de
    Validity
        Not Before: Aug  4 07:06:15 2014 GMT
        Not After : Sep  3 07:06:15 2014 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
            Public-Key: (1024 bit)
            Modulus:
                00:bf:93:1d:32:57:6b:49:b2:79:27:66:ff:44:f8:
                3a:60:e7:df:77:17:d7:79:69:e7:26:19:12:7a:ae:
                1d:9a:ac:8f:df:18:7d:13:dd:ce:b2:97:50:fb:5a:
                ea:8f:f5:53:ce:bf:35:d6:26:a0:ee:4f:2d:47:ca:
                52:13:c0:8a:63:f2:3c:bb:19:df:56:1e:f4:98:17:
                a7:7e:7c:31:aa:c7:8c:4d:58:a7:fc:d8:75:c0:28:
                d5:76:e5:4d:14:b6:99:58:b4:f2:b4:9f:2a:43:c2:
                49:cf:c3:2b:7d:a3:65:05:e7:d2:d9:8f:37:58:4e:
                e7:4c:29:ba:6f:4b:85:2f:c9
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Subject Key Identifier:
            CA:A1:3B:1A:CD:7D:98:43:1C:9F:7C:97:23:54:74:EF:B7:93:17:A1
        X509v3 Authority Key Identifier:
            keyid:CA:A1:3B:1A:CD:7D:98:43:1C:9F:7C:97:23:54:74:EF:B7:93:17:A1

        X509v3 Basic Constraints:
            CA:TRUE
  Signature Algorithm: sha1WithRSAEncryption
     1d:93:1a:a1:c9:f8:cb:c9:9b:55:81:97:6d:53:c2:93:d9:dd:
     dc:e1:c3:13:56:f9:7f:92:ca:a8:31:5c:c0:c4:20:95:07:dc:
     53:a9:a0:ff:9e:82:aa:35:34:5e:be:a1:59:f6:b0:dd:9b:35:
     4f:03:4f:93:37:96:b0:43:eb:7b:fc:a8:ff:32:32:8d:2b:20:
     bd:6f:09:1a:89:e8:b2:c7:18:c0:96:c1:74:1f:99:2e:de:f0:
     34:0a:c7:15:9b:a9:45:e5:ed:86:26:88:f3:35:45:2a:c5:cc:
     d3:a5:fa:a7:15:5d:d5:97:15:c8:0f:4f:23:01:5d:2a:3d:ba:
     4e:75
```

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.

Beschreiben Sie die Signatur des Herausgebers

Zusammenfassung

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

f. 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?

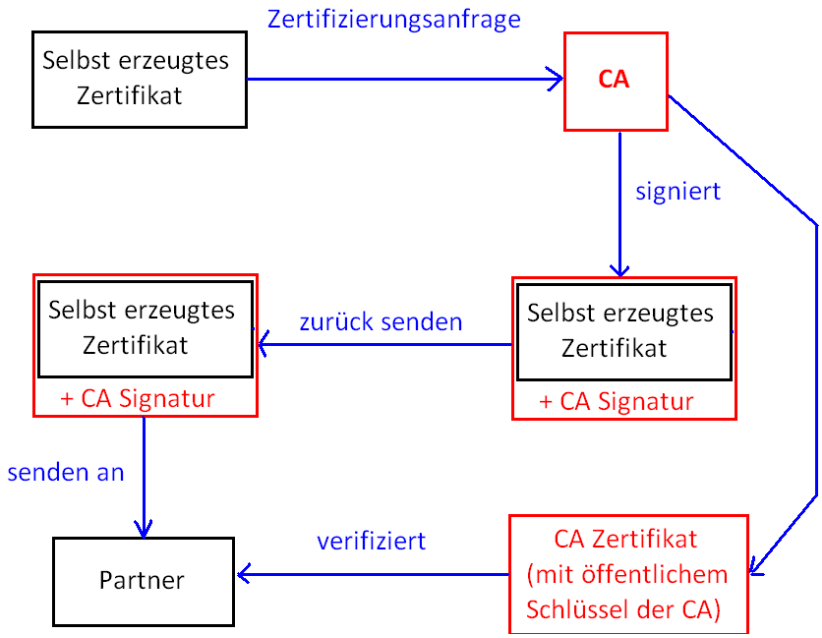
Im vorhergehenden Kapitel haben Sie gelernt, wie Sie Ihren eigenen öffentlichen Schlüssel in ein Zertifikat eingebracht haben und anschließend mit Ihrem eigenen privaten Schlüssel signiert haben, um ein selbstsigniertes Zertifikat zu erstellen.

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 nicht verändert werden kann, damit nicht ein Fremder seinen eigenen öffentlichen Schlüssel einbringt!

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.

Wenn Ihr Partner Ihren öffentlichen Schlüssel, von der CA signiert, 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 den öffentlichen Schlüssel der CA, mit dem die Signatur Ihres selbst-erzeugten Zertifikats verifiziert werden kann.

Erzeugen einer Zertifikatsignierungsanfrage (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\itsCA\user> openssl req -new -key hoefken_rsa_aes.key -out hoefken.csr

Enter pass phrase for hoefken_rsa_aes.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]: (return)
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\itsCA\user> type hoefken.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIBETCBvAIBADBXMQswCQYDVQQGEwJDTjELMAkGA1UECBMCUE4xCzAJBgNVBAcT
AkNOMQswCQYDVQQKEwJPTjELMAkGA1UECxMCVU4xFDASBgNVBAMTC0hlcm9uZyBZ
YW5nMFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKnmGcbuiAGG2XKek5LbVwF7AoT8
HuNXXip7KyWevbrFlSxJWSjfpmeGJo7/Nsw6hFwor28RyAy1wsW5BNYOXdECAwEA
AaAAMA0GCSqGSIb3DQEBBAUAA0EALE+d7H514HyQXu2CgwXYDvqZRngFLZFdGxQN
6AtEXXV+eC2c+URNBcmoF3oghJdPqZv7D1nZ7EBf20XSWzioQA==
-----END CERTIFICATE REQUEST-----
```

Hinweis: Es wurde die Datei *hoefken.csr* im Verzeichnis C:\OpenSSL\isCA\user erzeugt. Die Anfragedatei wird wieder im PEM Format gespeichert.

Anzeigen der Komponenten der Zertifikatsignierungsanfrage

Mit diesem Befehl können Sie die Komponenten der Zertifikatsignierungsanfrage anzeigen:

```
c:\OpenSSL\itsCA\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 (1024 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      :  unable to print attribute
    Signature Algorithm: sha1WithRSAEncryption
                2a:f6:cb:6e:1e:ae:7a:bf:b7:ff:41:34:3a:d8:d9:56:cc:f7:
                b0:c5:5b:6b:44:9d:dc:3c:f3:8d:e1:2d:bb:cd:3c:44:27:dd:
                51:74:3d:3a:c6:c9:b1:ac:3f:35:5f:0b:0e:33:bc:2e:11:f9:
                d2:9f:c3:b9:fa:ed:af:d6:99:1d:09:0d:0b:4f:11:b3:25:ee:
                bd:86:2c:8a:4a:68:54:b5:1c:16:9a:4d:36:4d:39:8b:3d:0a:
                eb:c3:bd:ef:16:e2:55:68:6d:45:5c:a4:84:a8:3d:9d:72:6b:
                77:cb:b8:e7:3c:8f:da:7d:4b:5f:10:e0:9f:dd:0c:4a:b1:48:
                a9:17
```

Einige interessante Anmerkungen:

- Die Anfrage ist mit dem 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 von der CA signiert wird. Dazu müssen Sie natürlich das Passwort der verschlüsselten CA-Schlüsseldatei kennen.

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

```
c:\OpenSSL\itsCA\user> openssl ca -in hoefken.csr -out hoefken.crt
Using configuration from C:\OpenSSL\bin\openssl.cfg
Loading 'screen' into random state - done
Enter pass phrase for c:/OpenSSL/isCA/private/cakey.pem:INFOSEC
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Aug  4 11:48:26 2014 GMT
            Not After : Aug  4 11:48:26 2015 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 Aug 4 11:48:26 2015 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/isCA/private/cakey.pem**). Es ist der private Schlüssel der CA. Sie sehen die Seriennummer des Zertifikats (1), die Gültigkeitsdauer (1 Jahr) und für wen dieses Zertifikat ist.

Das signierte Zertifikat wird im Verzeichnis C:\OpenSSL\isCA\newcerts gespeichert,

Schauen Sie sich einmal ihr 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 automatisch auf den neuesten Stand gebracht.

Das können Sie sich auch mit einem Texteditor ansehen.

Öffnen Sie die folgenden Dateien. Tragen Sie den aktuellen Wert von **serial** unten ein:

itsCA\serial Wert:

itsCA\index.txt

itsCA\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 Zertifikat nun noch in das PKCS#12-Format.

Kopieren Sie zuerst das signierte Zertifikat (Datei *c:\OpenSSL\itsCA\newcerts\01.pem)* nach *c:\OpenSSL\itsCA\user\hoefken.pem*.

```
c:\OpenSSL\itsCA\user> openssl pkcs12 -export -in hoefken.pem -inkey hoefken_rsa_aes.key -out
hoefken.p12
Loading 'screen' into random state - done
Enter pass phrase for hoefken_rsa_aes.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\itsCA\user* befinden sich jetzt folgende Dateien:

hoefken_rsa.key – erster erzeugter Schlüssel (nur für Demozwecke), kann gelöscht werden
hoefken_rsa_aes.key - der private und der public Key. Diese Datei darf nicht auf dem OpenSSL-Server bleiben und muss gut geschützt werden!
hoefken_rsa_aes_pub.key - der Public Key. Wird unter anderem auf dem OpenSSL-Server eingesetzt.
hoefken.csr - die Zertifikatsanforderung (certificate signing request). Kann gelöscht werden.
hoefken.crt - selbsterzeugtes Zertifikat im PEM-Format.
hoefken.pem - eigenes, von der CA signiertes Zertifikat im PEM-Format
hoefken.p12 - eigenes 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.

g. Zertifikatspfade und deren Prüfung

Dieser Abschnitt beschreibt:

- Was ist ein Zertifikatpfad?
- Zertifikatpfadprüfung
- Testen des Zertifikatspfads mit OpenSSL

Was ist ein Zertifikatspfad?

Zertifikatspfad (auch Zertifikatkette 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
```

Zertifikatspfadprüfung (Certification Path Validation)

Ein Zertifikatpfad 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.

i. 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\itsCA\user> openssl genrsa -des3 -out caadmin_rsa.key
...

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

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

Ergebnis: *schuba.crt*

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

Erzeugen der Zertifikatsignierunganforderung für Marko Schuba
c:\OpenSSL\itsCA\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\itsCA\user> openssl x509 -req -in schuba.csr -extfile c:\openssl\bin\openssl.cfg -extensions
v3_ca  -CA caadmin.crt -CAkey caadmin_rsa.key -out schuba.crt -set_serial 3
...
```

iii. **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!

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

Ergebnis: *nagel.crt*

Erzeugen des Schlüssels für Stefan Nagel

```
c:\OpenSSL\itsCA\user> openssl genrsa -des3 -out nagel_rsa.key
...
```

Erzeugen der Zertifikatsignierungsanforderung für Stefan Nagel

```
c:\OpenSSL\itsCA\user> openssl req -new -key nagel_rsa.key -out nagel.csr
...
```

Signierung von Stefan Nagel's Anfrage durch Marko Schubas Schlüssel

```
C:\OpenSSL\itsCA\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
```

v. Erzeugen Sie nun selber ein Zertifikat für Georg Hoever und signieren Sie es von Stefan Nagel.

4 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!

vi. Verifizieren des kürzesten Zertifikatpfads, nur ein Zertifikat:

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

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

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

c:\OpenSSL\itsCA\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).

vii. Verifizieren eines Zertifikatpfads mit zwei Zertifikaten:

```
C:\OpenSSL\itsCA\user> 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\itsCA\user> openssl verify -CAfile caadmin.crt hoever.crt
hoever.crt: /C=DE/ST=NRW/L=Aachen/O=FH Aachen/OU=ETIT/CN=Georg Hoever
error 20 at 0 depth lookup:unable to get local issuer certificate

C:\OpenSSL\itsCA\user> openssl verify -CAfile schuba.crt nagel.crt
strauch.crt: /C=DE/ST=NRW/L=Aachen/O=FH Aachen/OU=ETIT/CN=Stefan Nagel
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 0 (**hoever.crt**) unterbrochen. Kann den Herausgeber von **hoever.crt** nicht finden (s.o.).
- Test 3: Pfad an der Stelle 1 (**schuba.crt**) unterbrochen. Kann den Herausgeber von **schuba.crt** nicht finden.

viii. Verifizieren eines Zertifikatpfads von vielen Zertifikaten:

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

C:\OpenSSL\itsCA\user> openssl verify -CAfile caadmin.crt -untrusted nagel.crt hoever.crt
hoever.crt: /C=DE/ST=NRW/L=Aachen/O=FH Aachen/OU=ETIT/CN=Stefan Nagel
error 20 at 1 depth lookup:unable to get local issuer certificate

C:\OpenSSL\itsCA\user> copy schuba.crt + nagel.crt all.crt
C:\OpenSSL\itsCA\user> openssl verify -CAfile caadmin.crt -untrusted all.crt hoever.crt
hoever.crt: OK
```

Beachten Sie:

- Test 1: Perfekt, die Vertrauenskette ist bis zur Root-CA gegeben (nagel->schuba->caadmin).
- Test 2: Pfad an der Stelle 1 unterbrochen. Kann den Herausgeber von **nagel.crt** nicht finden.
- Test 3: Perfekt. Im Beispiel werden zwei Zertifikate mit dem DOS-Befehl "copy" zu einer Datei zusammengefügt.

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 gezeigten Befehle anwenden!

h. Erzeugung eines eigenen Zertifikats

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

Führen Sie alle dazu benötigten Befehle hier auf:

b. Schauen Sie sich die Komponenten Ihres Zertifikats an.

i. Welchen Befehl verwenden Sie dazu?

ii. Welche Seriennummer hat Ihr Zertifikat?

iii. Wer ist der Herausgeber Ihres Zertifikats?

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

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

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

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

Welches Export Passwort wollen Sie verwenden?

i. 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.

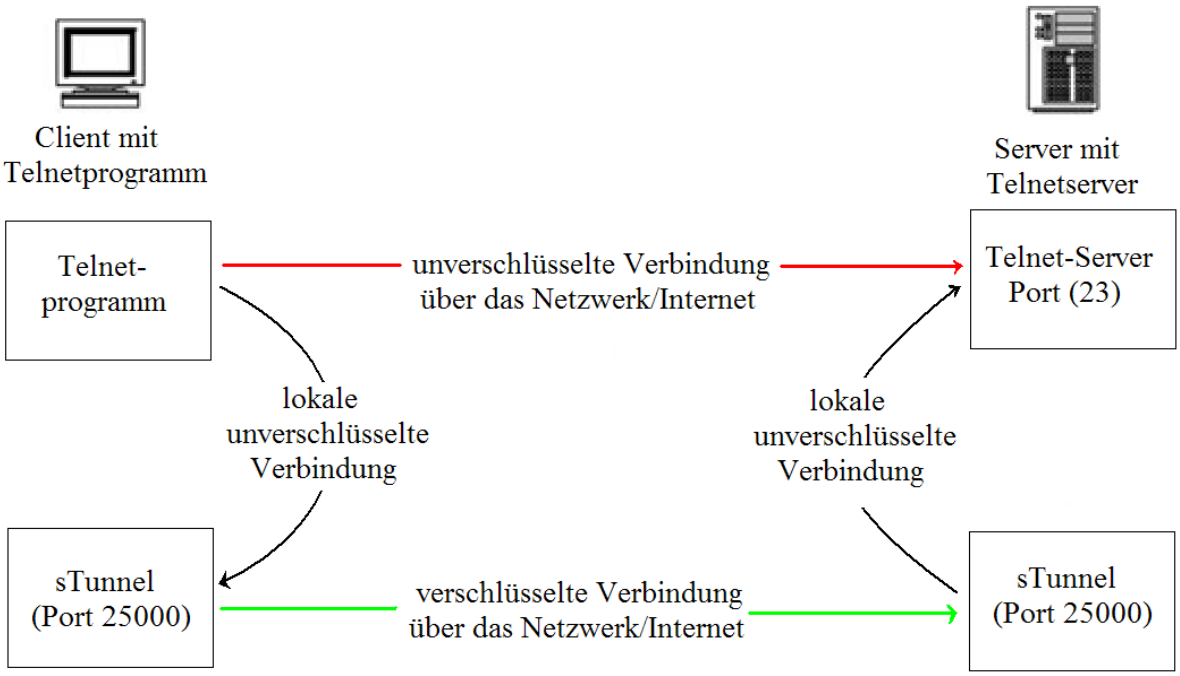
Starten Sie die **IS-Server1**.

Hinweis: Sollte die VM nicht vorhanden sein, verfahren Sie wie oben.

Starten Sie auf **Client1** und **Server1** die Datei **stunnel-4.21-installer.exe** (befindet sich entweder im Verzeichnis *c:\ITS-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 Sie das Programm **sTunnel** ein. Es soll eine Telnet Sitzung (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).

Konfigurationsdatei des Clients (infosec_cl.conf)

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

```
[telnet]
accept = 25000
connect = 10.0.0.3:25000  (Achtung: tragen Sie hier Ihre IP Adresse ein!)
```

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 Telnet Server 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.