

Virtuelle Private Netze - Handreichung zur Präsentation

VPN mit openvpn und openssl

Michael Dienert

8. Juli 2019

Inhaltsverzeichnis

1 Was ist ein VPN	2
1.1 Ein Beispiel	2
1.2 Definitionen und Begriffe	2
1.3 Unterschiedliche Varianten von VPNs	3
2 Rahmen, Pakete, virtuelle Verbindungen	3
2.1 Die beiden Schichtenmodelle der Netzwerktechnik	3
2.2 Virtuelle Verbindungen	4
2.3 Datentelegramme	4
2.4 Kapselung: mehrfache Verpackung der Daten	4
2.5 VPN mit Tunnel	5
3 Die virtuellen Netzwerkschnittstellen tun und tap	5
3.1 Physikalische Schnittstellen	5
3.2 Ethernet-Schnittstellen	5
3.3 Virtuelle Schnittstellen tun und tap	5
3.4 Virtuelle Tunnelschnittstelle im Schaubild	6
4 Asymmetrische Verschlüsselung - Public Key Kryptografie	7
4.1 Ein privater Brief	7
5 Eine superkurze Einführung zu RSA	7
5.1 RSA mit dem Numerikprogramm bc	7
6 Digitales Signieren mit RSA und md5 / sha	8
6.1 Digitales Signieren mit Hash-Algorithmen	8
6.2 Versuch: Hash-Werte mit openssl berechnen	8
6.3 Digitales Signieren mit Hash-Werten	9

7	Man-In-The-Middle-Angriff	9
7.1	Habe ich den richtigen öffentlichen Schlüssel?	9
7.2	Signierte öffentliche Schlüssel - Zertifikate	10
7.3	Wie wird ein Zertifikat erstellt	10
7.4	Analyse eines Zertifikats mit openssl	10
8	Ein Tunnelexperiment mit openvpn2	11
8.1	Wiederholung: Zertifikate	11
8.2	Erzeugung einer Public Key Infrastruktur	12
8.2.1	Schlüsselerzeugung mit easysrsa3	12
8.3	Installation der Schlüssel und Zertifikate	13
8.4	Installation von openvpn aus den Quellen	14
8.5	Analyse des Datenverkehrs durch den Tunnel mit tcpdump und nc . .	15

1 Was ist ein VPN

1.1 Ein Beispiel

- Eine Firma möchte die Netzwerke zweier Unternehmensstandorte miteinander verbinden.
- Beide Netze verwenden Adressen aus dem privaten Adressraum (z.B. 10.8.0.0/16)
- der Zugang ins Internet ist jeweils sehr restriktiv (Firewall)

klassische Lösung: Standleitung (Leased Line), arbeitet auf Layer 2.

- sicher, schnell, hohe Verfügbarkeit und Bandbreite
- teuer

Lösung mit VPN: die Daten, die die Standleitung transportieren würde, werden in IP-Pakete verpackt und durch das Internet geroutet.

- es wird keine physikalische Verbindung aufgebaut. Die Pakete können verschiedene Wege nehmen. ⇒ **virtuelle** Verbindung
- die Daten in den IP-Paketen werden verschlüsselt. ⇒ **private** Verbindung

1.2 Definitionen und Begriffe

Eine von vielen Definitionen:

Ein VPN ist eine verschlüsselte Verbindung zwischen zwei entfernten Segmenten eines einzigen, privaten Netzwerks über ein öffentliches Netzwerk, wie z.B. das Internet.

Site-to-Site VPN - zwei vollständige Netzwerke werden miteinander verbunden. Jede Seite ist mit einem VPN-Gateway ausgestattet.

Remote-Access VPN - Zugriff auf das Unternehmensnetzwerk durch Mitarbeiter von zu Hause aus. Auf dem entfernten Hostrechner muss eine VPN-Client-Software installiert sein.

1.3 Unterschiedliche Varianten von VPNs

Verschlüsselte VPNs - einfache Art eines VPNs: die Nutzdaten in den IP-Paketen, die durch das Internet wandern, werden verschlüsselt.

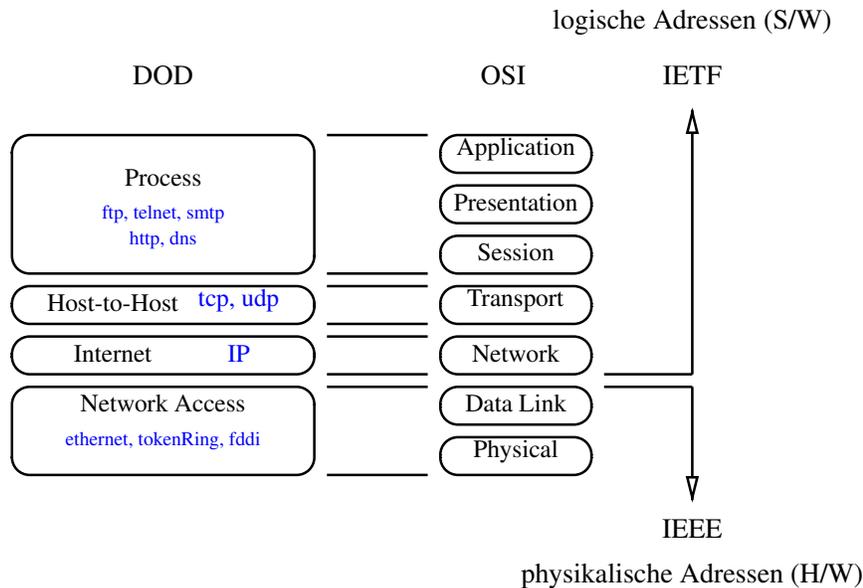
Die Bezeichnung *vpn* an dieser Stelle ist üblich, aber umstritten. Beispiele:

- ssh
- ssh mit Portweiterleitung (sehr interessant!)
- ppp über ssh
- ssl-vpn

Tunnelbasierte VPNs - häufigste Form eines VPNs: beim Tunneln werden komplette Pakete nochmals in ein IP-Paket verpackt (genauer gesagt: IP-Pakete werden in UDP/TCP-Segmente gesteckt, die dann wiederum in IP-Paketen reisen).

2 Rahmen, Pakete, virtuelle Verbindungen

2.1 Die beiden Schichtenmodelle der Netzwerktechnik



- Das Netz **zwischen** den lokalen (Campus-) - Netzen ⇒ Das **Inter** - Net

- Campus Berkeley: TokenRing, Campus Stanford: Ethernet II ⇒ unterschiedliche H/W ⇒ physikalische (MAC-) Adressen
- Das Inter-Net benötigt ein **Hardwareunabhängiges** Adressschema ⇒ 32-bit IP-Adresse ⇒ Routing wird möglich
- Daten werden **häppchenweise** geschickt ⇒ IP-Pakete

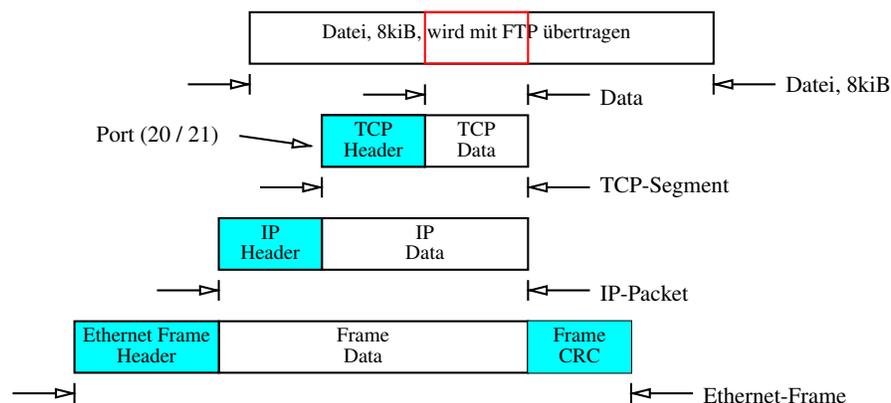
2.2 Virtuelle Verbindungen

- Problem: Anwendung möchte Bytestrom (wie Datei-I/O)
- TCP stellt **virtuelle** Verbindung her (Bytes durchnummeriert)
- Endpunkte der Verbindung: Sockets
 - mehrere Sockets pro Host
 - Socket über **Portnummer** identifiziert
 - bestimmte Portnummern werden Anwendungen zugeordnet (eingehende Verbindungen)

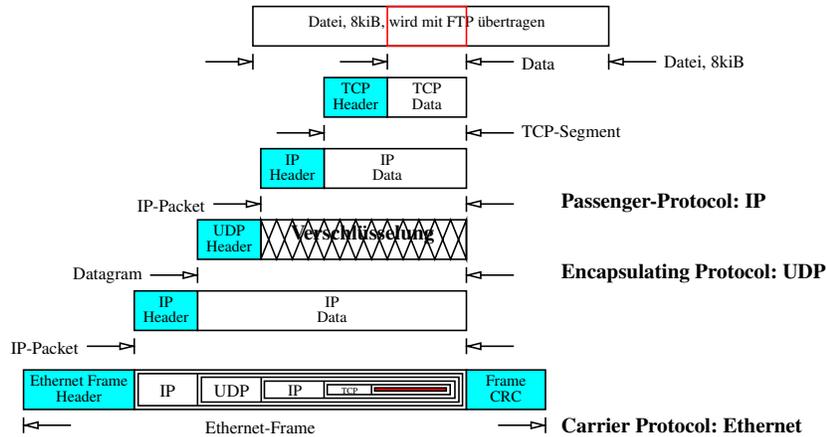
2.3 Datentelegramme

- Schwesterprotokoll: UDP
 - UDP arbeitet auch mit Portnummern
 - kein Bytestrom ⇒ bis 64kiB grosse **Datagramme** ⇒ wegschicken und vergessen
- deutlich schneller als TCP

2.4 Kapselung: mehrfache Verpackung der Daten



2.5 VPN mit Tunnel



3 Die virtuellen Netzwerkschnittstellen tun und tap

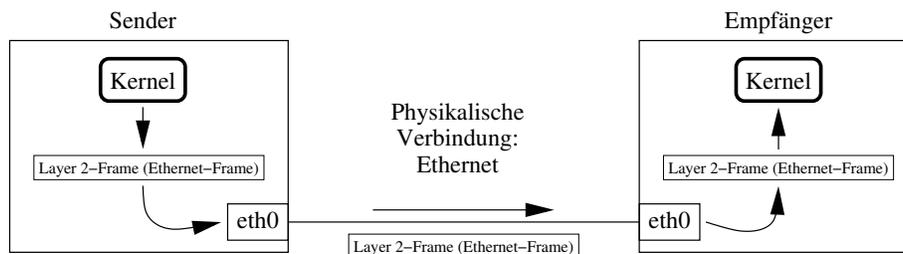
3.1 Physikalische Schnittstellen

- Die beiden Diagramme zur Kapselung und Tunnelung enden auf Layer 2
- Wie geht es unterhalb von Layer 2 weiter?
- Natürlich mit Layer 1, der physikalischen Schicht:
- Physikalische Ethernet-Schnittstellen auf unixoiden Rechnern heissen:

Linux: eth0, eth1, ...

bsd, osx: en0, en1, ...

3.2 Ethernet-Schnittstellen

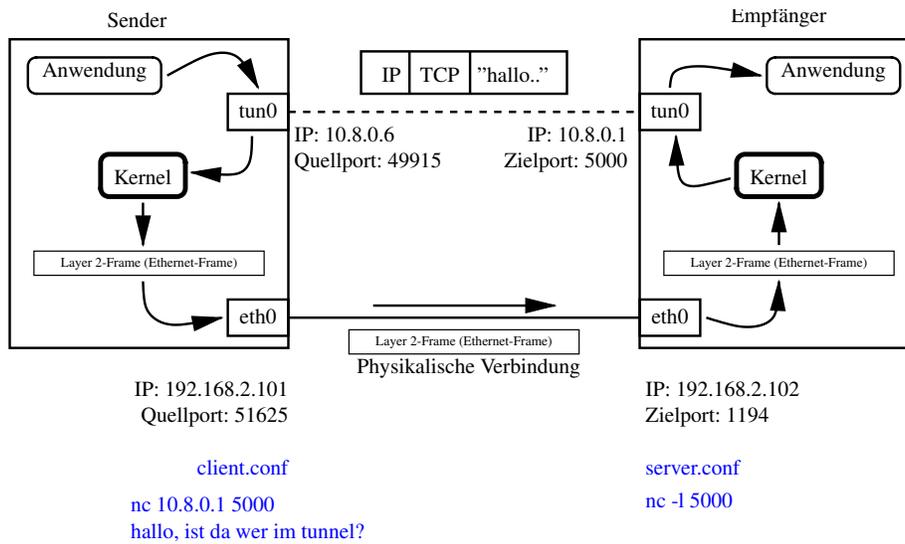


3.3 Virtuelle Schnittstellen tun und tap

tun - *tun* wird durch ein Kernelmodul (Treiber, Kernel-Extension) erzeugt. Man kann sich *tun* wie eine Netzwerkschnittstelle vorstellen, die Pakete auf Layer 3 (IP) über eine virtuelle Leitung sendet und empfängt.

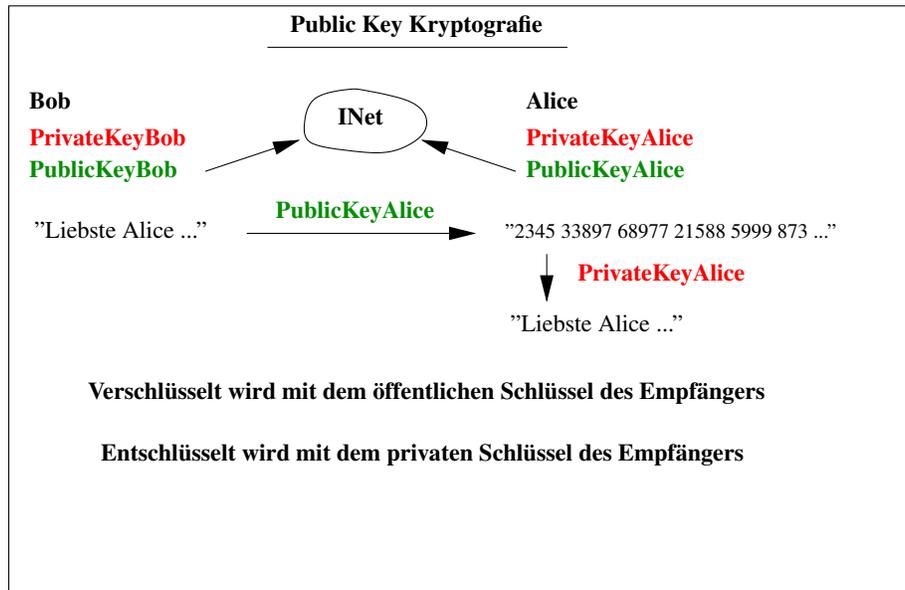
tap - *tap* ist ebenso eine virtuelle Schnittstelle, jedoch sendet und empfängt tap Layer 2 (Ethernet) Rahmen. Auf *tap* wird im Folgenden nicht weiter eingegangen.

3.4 Virtuelle Tunnelschnittstelle im Schaubild



4 Asymmetrische Verschlüsselung - Public Key Kryptografie

4.1 Ein privater Brief



5 Eine superkurze Einführung zu RSA

5.1 RSA mit dem Numerikprogramm bc

- *bc* ist ein Numerikprogramm mit beliebiger Auflösung für die Kommandozeile
- Ein RSA-Schlüsselpaar besteht aus 3 Zahlen (bis zu 1024 Bit Länge!)
 - Das Hauptmodul N , ein Produkt zweier grosser Primzahlen. N hat typ. 1024 Bit Länge
 - Dem Verschlüsselungsexponenten e = öffentlicher Schlüssel
 - Dem Entschlüsselungsexponenten d = privater Schlüssel
 - d und e lassen sich aus den Faktoren von N berechnen
 - Ganz wichtig: d und e sind austauschbar

Im Beispiel verschlüsseln wir den ASCII-Code von 'A' = 65 mit $N = 681$, $e = 151$, $d = 3$

- $Cif fre = 65^{151} \bmod 681 = 554$
- $Klartext = 554^3 \bmod 681 = 65$

Privater und öffentlicher Schlüssel vertauscht, $e = 3$, $d = 151$:

- $Ciffre = 65^3 \bmod 681 = 182$
- $Klartext = 182^{151} \bmod 681 = 65$

Führen Sie die Berechnung selbst mit dem Programm *bc* (Binary Calculator) durch! Hinweis: C-Syntax

6 Digitales Signieren mit RSA und md5 / sha

6.1 Digitales Signieren mit Hash-Algorithmen

Der bekannteste Hash-Algorithmus ist md5, was für *Message Digest Algorithmus No. 5* steht. Entwickelt wurde md5 von *Ron Rivest*, der zusammen mit *Adi Shamir* und *Leonard Adleman* auch den rsa-Algorithmus entwickelt hat.

sha steht für *Secure Hash Algorithm*, eine Entwicklung der NSA (National Security Agency). Publiziert wurde sha vom amerikanischen Standardisierungsinstitut NIST.

- md5 von Ron Rivest 1991 entwickelt
- sha von NSA (Nat. Security Agency) / NIST (Nat. Inst. of Standards and Technology)
- Hash Algorithmen berechnen aus beliebigen Eingangsdaten einen z.B 128bit - langen *Hash-Wert*

⇒ digitaler Fingerabdruck der Daten

6.2 Versuch: Hash-Werte mit openssl berechnen

openssl ist eine freie Sammlung von Dienstprogrammen fuer SSL und TLS. Es bringt für fast jedes Crypto-Problem eine Reihe von Werkzeugen mit.

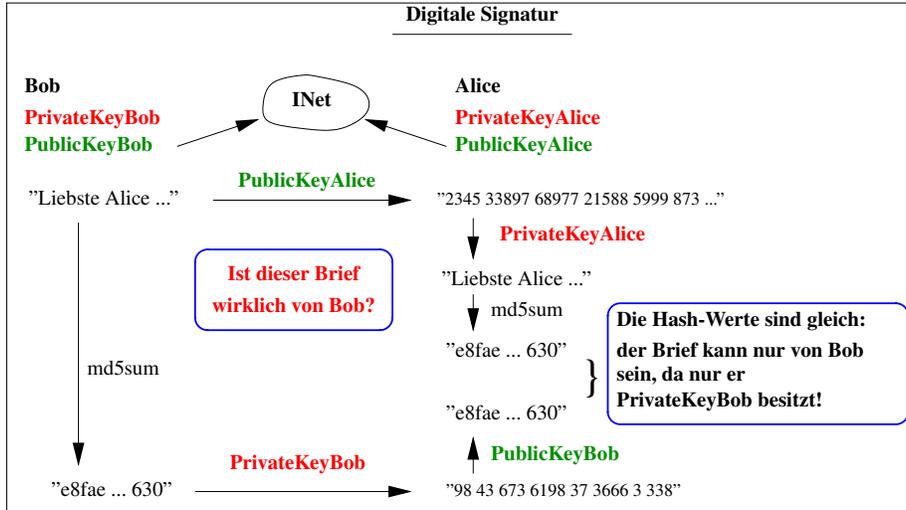
Am Rechner sollen nun mit openssl eine Reihe von Hash-Werten aus einer Textdatei berechnet werden. Dabei soll beobachtet werden, wie selbst kleine Änderungen am Dateitext zu einem total anderen Hash-Wert führen.

- openssl = Eierlegende-Crypto-Wollmilchsau
- unzählige hash-Algorithmen ⇒ man-Page lesen

```
micha@eddie:~$ man openssl  
  
micha@eddie:~$ openssl dgst -md5 <text-datei>
```

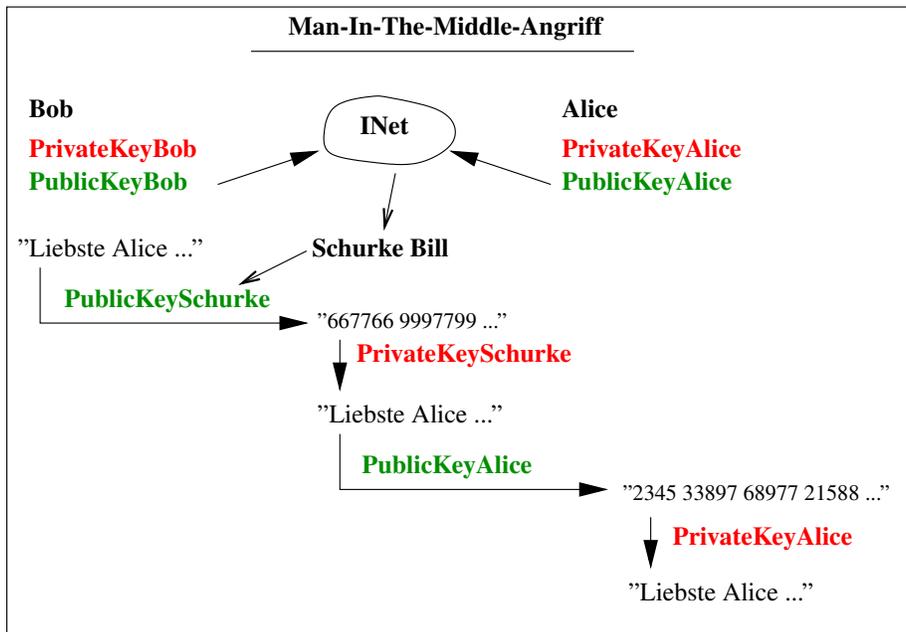
Winzige Änderungen an der Datei erzeugen einen total anderen Hash-Wert!

6.3 Digitales Signieren mit Hash-Werten



7 Man-In-The-Middle-Angriff

7.1 Habe ich den richtigen öffentlichen Schlüssel?



7.2 Signierte öffentliche Schlüssel - Zertifikate

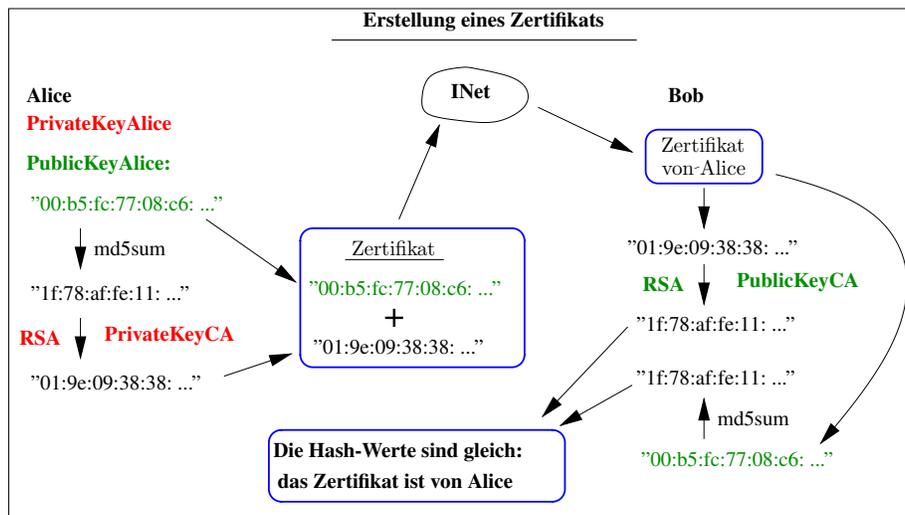
Problem: Bob muss sicher sein können, dass der angeforderte, öffentliche Schlüssel wirklich von Alice ist.

Lösung: Mit dem Schlüssel wird ein *Echtheitszertifikat* mitgeliefert, d.h. der `PublicKeyAlice` wird mit dem `PrivateKeyNotariat` signiert.

CA: Notariate heissen *Certificaton Authority* - CA

Bedingung: Die öffentlichen Schlüssel der CAs müssen vorab auf dem Rechner von Alice vorliegen, sonst wäre auch an dieser Stelle ein MITM-Angriff möglich.

7.3 Wie wird ein Zertifikat erstellt



7.4 Analyse eines Zertifikats mit openssl

```
micha@eddie:~$ openssl x509 -text -noout -in client.crt
```

```

Certificate:
Data:
  Version: 3 (0x2)
  Serial Number: 3 (0x3)
  Signature Algorithm: md5WithRSAEncryption
  Issuer: C=DE, ST=BW, L=FR, O=WARA, OU=IT, CN=MICHA/emailAddress=dienert@wara.de
  Validity
    Not Before: Jan 17 22:06:38 2010 GMT
    Not After : Jan 15 22:06:38 2020 GMT
  Subject: C=DE, ST=BW, O=WARA, OU=IT, CN=client/emailAddress=dienert@wara.de
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
      Modulus (1024 bit):
        00:a9:50:c0:1e:e4:93:ce:ec:4c:bf:3b:c8:14:a1:
        62:ae:c8:2c:6c:f1:78:3d:21:36:4f:45:6e:d3:ab:
        90:99:4c:25:bf:e7:0d:91:ec:83:15:a8:b4:71:99:
        3c:a5:28:a1:93:6b:a7:34:74:4a:c3:ad:4b:e3:7a:
        2a:97:cf:8b:15:cc:f9:f1:9a:e9:0b:3a:ea:c2:4b:
        40:8f:de:50:30:4d:d2:f8:6c:8b:0c:ff:c4:7a:31:
        9c:2b:a7:41:df:13:0c:cf:aa:64:be:c3:67:44:f8:
        22:b3:79:4d:bb:5c:95:fa:79:ad:cf:3a:05:79:3b:
        88:8e:ad:fe:2c:59:4f:f9:81
      Exponent: 65537 (0x10001)

```

```

X509v3 extensions:
  X509v3 Basic Constraints:
    CA:FALSE
  Netscape Comment:
    OpenSSL Generated Certificate
  X509v3 Subject Key Identifier:
    5C:9C:53:A7:46:E2:49:91:BF:F9:9C:69:0E:4E:5E:32:0C:E7:C1:F3
  X509v3 Authority Key Identifier:
    keyid:44:80:03:E9:3F:24:F0:0E:30:04:16:69:6A:7A:2F:A1:C5:5E:46:46
    DirName:/C=DE/ST=BW/L=FR/O=WARA/OU=IT/CN=MICHA/emailAddress=dienert@wara.de
    serial:C6:8A:A7:C8:B1:E2:DD:EF

Signature Algorithm: md5WithRSAEncryption
8a:c6:82:a0:48:59:2e:7f:55:f3:16:fa:0e:68:4c:11:d7:56:
0b:d3:85:0b:da:22:e1:3b:d6:60:f7:0b:06:19:6f:11:1d:ca:
c0:b9:1d:ff:d1:d1:7e:7f:97:63:89:04:18:93:cb:9a:09:06:
aa:0a:b2:b5:88:de:c7:ec:db:49:93:1b:bc:13:cc:05:69:c9:
ba:f7:92:67:2f:fd:e0:c1:98:6c:23:b9:38:39:17:f5:d7:6e:
1f:70:58:79:1b:cb:d7:0d:7f:d9:5d:77:b9:3d:9e:90:80:67:
54:be:f6:41:49:95:65:ed:67:81:84:2a:a8:19:65:91:4b:eb:
5a:41

```

8 Ein Tunnelexperiment mit openvpn2

8.1 Wiederholung: Zertifikate

- Zwischen Server und Client wird eine verschlüsselte Verbindung mit SSL eingerichtet
- Server und Client benötigen dazu *signierte, öffentliche* Schlüssel: ≡ **Zertifikate**
- Und natürlich die zugehörigen *privaten* Schlüssel.
- Wir werden mit *selbstsignierten* Schlüsseln arbeiten. Dazu muss eine eigene *Certification Authority* aufgebaut werden.
- Um die privaten Schlüssel und die Zertifikate von CA, Client und Server zu erzeugen, wird das Programm `easyrsa3` verwendet.

```
https://github.com/OpenVPN/easy-rsa.git
```

8.2 Erzeugung einer Public Key Infrastruktur

Unbedingt beachten:

Beim Erzeugen, Installieren oder Verteilen von Schlüsseln darf der private Schlüssel (erkennbar an der Dateiendung `*.key`) **niemals über einen unsicheren Kanal übertragen werden.**

Optimal ist, wenn der private Schlüssel dort erzeugt wird, wo er benötigt wird. Dann ist eine Übertragung nicht notwendig.

- In der Praxis sind an der Schlüsselerzeugung drei Instanzen beteiligt:
 1. die Certification Authority - CA
 2. der VPN-Server
 3. der VPN-Client
- Server und Client erzeugen je ein Schlüsselpaar
- Der öffentliche Schlüssel wird zur CA geschickt und von dieser signiert.
- Die CA schickt die so entstandenen Zertifikate (client.crt, server.crt) zurück
- Die privaten Schlüssel bleiben wo sie sind (je auf Server und Client)

8.2.1 Schlüsselerzeugung mit easyrsa3

- Aufgabe: mit dem Programm `easyrsa3` sollen die für openVPN nötigen Schlüssel erzeugt werden.
- Man kann dazu 3 verschiedene Rechner verwenden
- Oder man erzeugt 3 verschiedene Verzeichnisse auf einem Rechner
- Folgendes How-To zeigt die Erzeugung auf einem einzelnen Rechner

```
cd ~
git clone https://github.com/OpenVPN/easy-rsa.git
cd easy-rsa/
cp -a easyrsa3/ easyrsa3Server
cp -a easyrsa3/ easyrsa3Client

mv easyrsa3 easyrsa3CA
cd easyrsa3CA
./easyrsa init-pki
./easyrsa build-ca nopass
# common-name beliebig

cd ../easyrsa3Client/
./easyrsa init-pki
./easyrsa gen-req client nopass
# common-name muss 'client' heissen
```

```

cd ../easysrsa3Server/
./easysrsa init-pki
./easysrsa gen-req server nopass
# common-name muss 'server' heissen
./easysrsa gen-dh

cd ../easysrsa3CA
./easysrsa import-req ../easysrsa3Client/pki/reqs/client.req client
./easysrsa import-req ../easysrsa3Server/pki/reqs/server.req server

./easysrsa sign client client
./easysrsa sign server server

openssl x509 -text -noout -in pki/issued/client.crt
openssl x509 -text -noout -in pki/issued/server.crt

```

Der Inhalt des Verzeichnisses *easysrsa3CA/pki* müsste jetzt so aussehen:

```

.
|-- ca.crt
|-- certs_by_serial
|  |-- 01.pem
|  |-- 02.pem
|-- index.txt
|-- index.txt.attr
|-- index.txt.attr.old
|-- index.txt.old
|-- issued
|  |-- client.crt
|  |-- server.crt
|-- private
|  |-- ca.key
|-- reqs
|  |-- client.req
|  |-- server.req
|-- serial
|-- serial.old

```

Der Inhalt der pki-Verzeichnisse von *easysrsa3Server* und *easysrsa3Client* ist einfacher:

```

.
|-- dh.pem
|-- private
|  |-- server.key
|-- reqs
|  |-- server.req

.
|-- private
|  |-- client.key
|-- reqs
|  |-- client.req

```

8.3 Installation der Schlüssel und Zertifikate

- Das Verzeichnis `~/server` Alles erzeugen

- Das Verzeichnis `~/clientAlles` erzeugen
- Die erzeugten Schlüssel in die Verzeichnisse `serverAlles/` und `clientAlles/` kopieren
- Diese beiden Verzeichnisse müssen dann wie folgt aussehen. (Die Dateien `server.conf` und `client.conf` werden nach der Installation des `openvpn`-Quellpakets erzeugt):

```
serverAlles
|-- ca.crt
|-- dh.pem
|-- server.conf
|-- server.crt
`-- server.key

clientAlles
|-- ca.crt
|-- client.conf
|-- client.crt
`-- client.key
```

8.4 Installation von `openvpn` aus den Quellen

Die Installation von `openvpn` erfolgt direkt aus den Quellen. Dazu das Quellpaket ins Heimatverzeichnis (`~`) herunterladen und den Rest in einer Konsole ausführen. Hierzu sind `root`-Rechte notwendig:

```
cd ~
wget https://swupdate.openvpn.org/community/\
      releases/openvpn-2.3.8.tar.gz
tar -xzf openvpn-2.3.8.tar.gz
cd openvpn-2.3.8
./configure
make
make install
```

Folgende, abschliessende Schritte sind notwendig:

- Kopieren Sie die Datei `server.conf` aus dem Verzeichnis `sample-config-files/` nach `serverAlles/`
- Kopieren Sie die Datei `client.conf` aus dem Verzeichnis `sample-config-files/` nach `clientAlles/`
- In `server.config` muss man den Namen der Diffie-Hellman-Parameterdatei nach `dh.pem` ändern, in `client.config` müssen Sie die **Remote-Adresse** anpassen.

Die öffentlichen Schlüssel in `serverAlles` und `clientAlles` sind nun mit dem gleichen, privaten Schlüssel der *Certification Authority*, dem `ca.key` signiert. Aus Gründen der Einfachheit haben wir das alles auf *einer* Maschine gemacht. Das bedeutet aber nun, dass das die entsprechenden Verzeichnisse (`serverAlles`, `clientAlles`) auf den entsprechenden Host kopiert werden müssen.

Z.B. können wir das Verzeichnis `clientAlles` mit `scp` auf einen entfernten Host kopieren:

```
scp -r ~/clientAlles root@r025-99:~
```

Jetzt können Server und Client gestartet werden. Da `openvpn` die Schlüssel und Konfigurationen **relativ zum aktuellen Verzeichnis** sucht, müssen Sie das Kommando **innerhalb** von `serverAlles/` bzw. `clientAlles/` ausführen: Im Verzeichnis `serverAlles/` auf dem Server-Host:

```
openvpn server.conf
```

Im Verzeichnis `clientAlles/` auf dem Client-Host:

```
openvpn client.conf
```

8.5 Analyse des Datenverkehrs durch den Tunnel mit `tcpdump` und `nc`

Damit der Tunnelverkehr lesbar, also unverschlüsselt übertragen wird, tragen Sie bitte in `server.conf` und `client.conf` folgende Zeile ein und schalten die `lzo`-Komprimierung aus:

```
cipher none  
;comp-lzo
```

Mit dem Kommando `tcpdump` kann nun der Datenverkehr an der physikalischen Schnittstelle mitgeschnitten werden:

```
tcpdump -nti eth0 -XX port 1194
```

Damit nun Daten durch den Tunnel geschickt werden, verwenden wir das Programm `netcat`. Und zwar auf der Client **und** auf der Serverseite (auf BSD-Systemen muss man den Serverport mit dem Schalter `-p` angeben, unter Linux nicht): Auf dem Server:

```
nc -l 5000 (bsd: nc -l -p 5000)
```

Auf dem Client:

```
nc 10.8.0.1 5000  
hallo, ist da wer im tunnel?
```

Und so soll es aussehen:

```
IP 192.168.2.101.51625 > 192.168.2.102.1194: UDP, length 107  
0x0000: 001b 63fe 8862 001c b3c3 830f 0800 4500 ..c..b.....E.  
0x0010: 0087 5602 0000 4011 9e48 c0a8 0265 c0a8 ..V...@..H...e..  
0x0020: 0266 c9a9 04aa 0073 7931 30c6 0974 9156 .f.....sy10..t.V  
0x0030: 1912 07dc f814 0bb3 c19e b31f c5c0 2200 ....E..Q(%@.@..k  
0x0040: 0000 f8fa 4500 0051 2825 4000 4006 fe6b ....E..Q(%@.@..k  
0x0050: 0a08 0006 0a08 0001 c2fb 1388 17a9 3e1b .....>.....  
0x0060: 130d b2a4 8018 ffff aldc 0000 0101 080a .....>.....  
0x0070: 0573 769c 24ad 0718 6861 6c6c 6f2c 2069 .sv.$...hallo,i  
0x0080: 7374 2064 6120 7765 7220 696d 2074 756e st.da.wer.im.tun  
0x0090: 6e65 6c3f 0a nel?.
```