

Kryptographie

- SSH

- SSH-Keys erzeugen und verwalten
- Mit der SSH-Passphrase an KDE/Gnome anmelden
- offending key aus known_hosts löschen
- SFTP mit OpenSSH Boardmitteln
- SSH als SocksProxy
- SSH-Clientkonfiguration Multiplexing
- SSH-Forwarding über SUDO behalten
- SSH-Keys von OpenSSH nach SSH2 konvertieren und zurück
- SSH-Tunnel

- GPG / GnuPG / PGP

- GPG / GnuPG / PGP Cheat-Sheet
- PGP public key von Keyserver holen und exportieren

- TLS / SSL

- OpenSSL - TLS-/SSL-Zertifikate
- Diffie-Hellman Parameterdatei Bitgröße ermitteln
- (Zwischen-)Zertifikat Key und Zertifikatsrequest per Script auf Plausibilität prüfen
- CAcert Root Zertifikat unter Debian/Ubuntu einbinden
- Signatur-Algorithmen einer Zertifikatskette anzeigen

- VPN (Wireguard / OpenVPN / IPSec)

- OpenVPN nach Hause
- IPSec Roadwarrior-VPN via racoon
- Tunnelüberwachung via Script
- OpenVPN startet nicht (ca md too weak)

SSH

SSH-Keys erzeugen und verwalten

SSH-Key erzeugen

Shortcut: `ssh-keygen -t ed25519 -a 100 -C 'email@mydomain.de'`

Es wird eine Passphrase abgefragt. Soll das Keyfile unverschlüsselt gespeichert werden, kann die Eingabe eines Passworts mit Enter übersprungen werden. Dies ist allerdings nicht empfehlenswert, denn falls der Key in die falschen Hände gelangt, wären alle Server mit dem Pubkey in den `authorized_keys` ohne weitere Passwortabfrage offen.

```
ssh-keygen -t [ (dsa) | (ecdsa) | ed25519 | rsa | (rsa1) ] -b 4096 -a 100 -C '<kommentar>'
```

als copy&paste Beispiel mit ed25519 (hat eine fixe Bitgröße von 256 bit,

sie muss deshalb nicht angegeben werden):

```
ssh-keygen -t ed25519 -a 100 -C 'email@mydomain.de'
```

Dies erzeugt zwei Dateien in `~/.ssh/id_<key_type>*` je nach verwendetem Verfahren. Beispiel RSA: in der Datei `id_rsa` steht der (verschlüsselte) private Key. In der Datei `id_rsa.pub` steht der öffentliche Teil des Schlüssels, der verteilt werden darf/muss.

Der Typ `ed25519` ist zum heutigen Stand (05.03.2021) die beste Wahl. Allerdings wird er noch nicht überall unterstützt. In diesem Fall empfiehlt es sich (zusätzlich) einen RSA-Key mit 4096 bit zu erstellen.

`ecdsa` sollte nicht verwendet werden, da möglicherweise eine Hintertür für die US-Regierung eingebaut ist. `RSA1` und DSA sollten auch nicht mehr verwendet werden.

Unter Windows gibts mit `puttygen.exe` ein grafisches Pendant dazu. Hier lassen sich die erzeugten Keys auch ins Unix-Format konvertieren.

Passphrase des SSH-Keys nachträglich ändern

```
ssh-keygen -p -f ~/.ssh/id_rsa
```

SSH-Key in den SSH-Agenten laden

Der SSH-Agent muss gestartet sein, bei Fedora Core passiert dies automatisch beim Starten der X-Oberfläche.

```
ssh-add
```

Dies lädt defaultmäßig alle `id_*`-Files in `~/.ssh` in den Agent. Der Agent dient der Bequemlichkeit, d.h. man lädt einmal seinen Key und von nun an kümmert sich der Agent um alle anfragenden Programme, wie z.B. `ssh` oder `scp`. Für Windows gibts im Putty-Paket ein Programm namens `pagent.exe`, welches den gleichen Zweck erfüllt.

Hier ist eine Anleitung, wie man seinen `ssh`-Key für die Anmeldung an KDE/Gnome verwenden und ihn gleich in den Agent laden kann.

SSH-Pubkey verteilen

Um nun den SSH-Key für die Authentifizierung an anderen Servern verwenden zu können muss er noch dort abgelegt werden. Dazu bringt `openssh` ein schönes Tool mit:

```
ssh-copy-id [user@]machine
```

Dies erzeugt auf dem Zielsystem ggf. das Verzeichnis `~/.ssh`, legt dort den öffentlichen Teil des Schlüssels in der Datei `~/.ssh/authorized_keys` ab und vergibt gleich passende Berechtigungen.

Fingerprint des SSH-Keys anzeigen

Mit dem Fingerprint lässt sich der Key schnell von anderen verifizieren (könnte z.B. telefonisch abgeglichen werden).

```
# Fingerprint als SHA256 ausgeben (Default bei Debian 8.8)
```

```
ssh-keygen -l -f ~/.ssh/id_rsa.pub
```

```
# Fingerprint als MD5 ausgeben
```

```
ssh-keygen -l -E md5 -f ~/.ssh/id_rsa.pub
```

Mit der SSH-Passphrase an KDE/Gnome anmelden

Dieses Dokument beschreibt das Einloggen in KDE/Gnome mit der SSH-Passphrase unter Fedora Core (6).

Ein vorhandenes Schlüsselpaar an den bekannten Orten (`~/.ssh/id_rsa` `~/.ssh/id_rsa.pub`) wird vorausgesetzt.

Zuerst muss das `pam_ssh` Paket nachinstalliert werden (als root):

```
yum -y install pam_ssh
```

nun muss noch die PAM-Konfiguration angepasst werden (als root):

```
cd /etc/pam.d
cp system-auth system-auth-ssh
```

jetzt die Datei `system-auth-ssh` anpassen:

```
##PAM-1.0
# This file is auto-generated.
# User changes will be destroyed the next time authconfig is run.
auth      required      pam_env.so
auth      sufficient     pam_ssh.so
auth      sufficient     pam_unix.so nullok try_first_pass
auth      requisite      pam_succeed_if.so uid >= 500 quiet
auth      required      pam_deny.so

account    required      pam_unix.so
account    sufficient     pam_succeed_if.so uid < 500 quiet
account    required      pam_permit.so

password   requisite      pam_cracklib.so try_first_pass retry=3
password   sufficient     pam_unix.so md5 shadow nullok try_first_pass use_authok
password   required      pam_deny.so

session    optional      pam_keyinit.so revoke
session    required      pam_limits.so
session    [success=1 default=ignore] pam_succeed_if.so service in crond quiet use_uid
session    required      pam_unix.so
session    optional      pam_ssh.so
```

die Datei `/etc/pam.d/kdm`

```
##PAM-1.0
auth      include        system-auth-ssh
account    required      pam_nologin.so
account    include        system-auth-ssh
password   include        system-auth-ssh
session    include        system-auth-ssh
session    required      pam_loginuid.so
session    optional      pam_selinux.so
session    optional      pam_console.so
```

die Datei `/etc/pam.d/gdm`

```
##PAM-1.0
auth      required    pam_env.so
auth      include     system-auth-ssh
account   required    pam_nologin.so
account   include     system-auth-ssh
password  include     system-auth-ssh
session   optional    pam_keyinit.so force revoke
session   include     system-auth-ssh
session   required    pam_loginuid.so
session   optional    pam_console.so
```

jetzt die X-Session neu starten. Nun kann man sich mit seiner SSH-Passphrase oder dem normalen Unix-Kennwort (nur wenn sich die Kennwörter unterscheiden) anmelden. Falls man sich mit der SSH-Passphrase anmeldet wird auch gleich der SSH-Agent gestartet und der Key geladen.

offending key aus known_hosts löschen

Welcher Linux-Admin kennt es nicht. Ein Server wurde neu installiert oder der SSH-Hostkey hat sich geändert und jetzt wird beim Versuch sich einzuloggen folgende Meldung angezeigt:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
1a:d4:e6:4f:1a:4f:61:ef:bc:b8:37:a3:22:d9:70:40.
Please contact your system administrator.
Add correct host key in /home/user/.ssh/known_hosts to get rid of this message.
Offending key in /home/user/.ssh/known_hosts:898
RSA host key for meinserver has changed and you have requested strict checking.
Host key verification failed.
```

Um sich nun wieder mit dem Ziel verbinden zu können, muss der alte Hostkey aus der known_hosts Datei entfernt werden. Die einfachste Methode ist mit einem kleinen Helper-Script:

```
#!/bin/bash
# 2010 Oliver Voelker <code@magenbrot.net>
#
# delete the given line number from ~/.ssh/known_hosts
#
if [ ${1} -gt 0 ]; then
    echo "Deleting line ${1} from ~/.ssh/known_hosts"
    sed -i "${1}d" ~/.ssh/known_hosts
else
    echo "Clear <line> from ~/.ssh/known_hosts"
    echo "Usage: ${0} <line>"
fi

exit 0
```

Einfach in ~/bin oder /usr/local/bin ablegen. Beim Aufruf des Scripts einfach die angemerkte Zeilennummer angeben (z.B. „ck 898“) und die betreffende Zeile wird aus der Datei gelöscht.

Sollte euer Sed keine -i Option besitzen, kann die Zeile etwa auch durch Perl entfernt werden:

```
perl -ni -e 'print if ($. != 898);' ~/.ssh/known_hosts
```

Alternativ über den VI-Editor:

```
vi +898d +x ~/.ssh/known_hosts
```

und es gibt sicherlich noch viele weitere Methoden da draußen ;)

SSH

SFTP mit OpenSSH Boardmitteln

Diese Anleitung wurde mit Debian Jessie und OpenSSH Version 6.7p1 getestet.

Folgender Teil wird am Ende der SSHd Konfiguration eingefügt:

```
# chroot for group 'sftponly' and individual users
Match Group sftponly
  ChrootDirectory %h
  AuthorizedKeysFile /etc/ssh/authorized_keys/%u
  ForceCommand internal-sftp
  AllowTcpForwarding no
  PermitTunnel no
  X11Forwarding no
```

danach den SSH daemon neu starten mit `systemctl restart ssh.service`.

Nun wird eine Gruppe angelegt, jedes Mitglied dieser Gruppe bekommt obige Einstellungen beim Login per SSH/SFTP zugewiesen: `groupadd sftponly`

Die SFTP-only Benutzer werden folgendermaßen angelegt:

```
useradd -g www-data -G sftponly -m -s /bin/false testuser
(optional:) passwd testuser
(oder per Pubkey:) vi /etc/ssh/authorized_keys/testuser
chown root:root /home/testuser
mkdir /home/testuser/exchange
chown -R testuser /home/testuser/exchange
```

Da für SFTP das Homeverzeichnis root:root gehören muss, empfiehlt es sich die Userdaten in ein Unterverzeichnis zu verlegen („exchange“ in diesem Fall).

SSH als SocksProxy

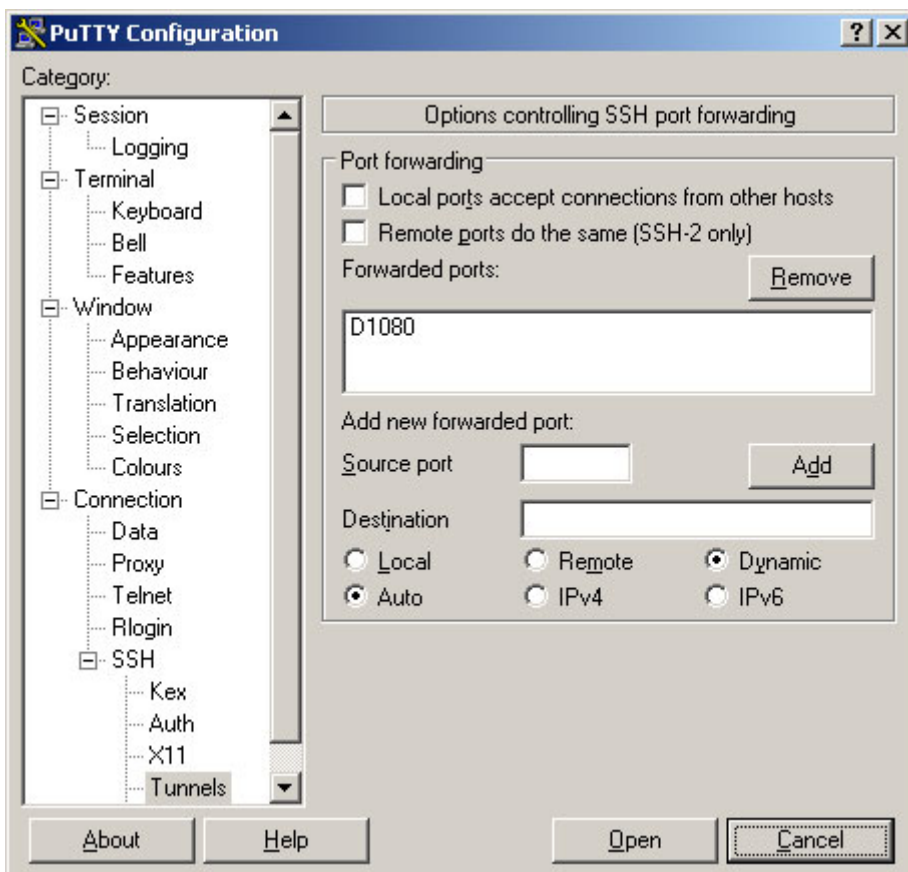
Sinn eines SocksProxy ist es z.B. auf Server und Dienste innerhalb eines Firmennetzwerks zugreifen zu können ohne jedoch dafür Ports in der Firewall aufreissen zu müssen oder auch um Netzwerkverkehr verschlüsselt über einen anderen PC zu tunneln.

SSH Tunnel unter Linux

```
ssh -N -D1080 username@remotehost.de
```

SSH Tunnel unter Windows (mit Putty)

Putty starten und ganz normal unter Hostname den Server eintragen über den getunnelt werden soll. Dann in den Optionen unter Tunnels unter forwarded Ports z.B. Port 1080 eintragen und den Button „Dynamic“ aktivieren.



Anwendungen auf das Socks-Protokoll umstellen

Manche Anwendungen wie z.B. Firefox oder manche Chat-Programme wie ICQ unterstützen das SOCKS-Protokoll von sich aus. Hierzu muss in den Optionen als Socks-Hosts der „localhost“ mit Port 1080 und SOCKS-Version 5 eingetragen werden.

Anwendung ohne SOCKS-Unterstützung

Jedoch unterstützen nicht alle Anwendungen das SOCKS-Protokoll. Für dieses Problem gibts jedoch sogenannte Socks-Clients wie z.B. [<http://www.freecap.ru/eng/>]

SSH-Clientkonfiguration Multiplexing

Hier meine Clientkonfiguration. Das Snippet kann entweder global in `/etc/ssh/ssh_config` oder im Home `~/.ssh/config` hinterlegt werden. Diese Einstellungen sorgen dafür, dass die Verbindung länger bestehen bleibt (Keepalive) und eine SSH-Verbindung mehrfach benutzt werden kann. Weitere Logins auf dem Zielsystem sind damit deutlich schneller.

```
Host *
  ForwardAgent no
  ForwardX11 no
  ForwardX11Trusted no
  StrictHostKeyChecking ask
  TCPKeepAlive=yes
  ServerAliveInterval=15
  ServerAliveCountMax=6
  Compression=yes
  ControlMaster auto
  ControlPath /tmp/%r@%h:%p
  ControlPersist yes
```

SSH-Forwarding über SUDO behalten

Aus Sicherheitsgründen werden beim Wechsel per Sudo auf einen anderen Benutzer die Umgebungsvariablen (Environment) gelöscht. Allerdings kann es nützlich sein, die Variable `SSH_AUTH_SOCK` zu behalten, damit das SSH-Forwarding weiter funktioniert.

Im SSH-Client muss natürlich das Agentforwarding aktiviert sein (global in `/etc/ssh/ssh_config` oder in `~/.ssh/config`):

```
Host *  
    ForwardAgent Yes
```

Dann folgende Zeile unter dem „Defaults env_reset“ eintragen (visudo):

```
Defaults    env_reset  
Defaults    env_keep+=SSH_AUTH_SOCK
```

Das funktioniert allerdings nur, wenn man von einem normalen User auf root wechselt, da der Auth-Socket nur vom ursprünglichen Benutzer lesbar ist (root darf aber natürlich alles):

```
user@server:/$ ssh-add -l  
4096 34:5b:42:b6:6f:f7:28:3e:54:e9:76:14:43:a2:04:c5 user@server (RSA)  
user@server:~$ sudo su  
root@server:/# ssh-add -l  
4096 34:5b:42:b6:6f:f7:28:3e:54:e9:76:14:43:a2:04:c5 user@server (RSA)
```

Unter Debian Wheezy hat es per „sudo su -“ nicht mehr funktioniert. Allerdings geht es mit „sudo -i“:

```
user@server:~$ ssh-add -l  
4096 34:5b:42:b6:6f:f7:28:3e:54:e9:76:14:43:a2:04:c5 user@server (RSA)  
user@server:~$ sudo -i  
root@server:~# ssh-add -l  
4096 34:5b:42:b6:6f:f7:28:3e:54:e9:76:14:43:a2:04:c5 user@server (RSA)
```

Dies wurde mit Debian/Ubuntu getestet, sollte aber auch in anderen Distributionen funktionieren.

SSH

SSH-Keys von OpenSSH nach SSH2 konvertieren und zurück

Hier wird gezeigt, wie SSH-Keys zwischen verschiedenen Typen von SSH-Servern konvertiert werden können. Dazu wird eine OpenSSH-Installation benötigt (Paket openssh-client bei Debian).

SSH2-Key zu OpenSSH-Key konvertieren:

```
# ssh-keygen -i -f ~/.ssh/id_dsa_1024_a.pub > ~/.ssh/id_dsa_1024_a_openssh.pub
```

OpenSSH-Key zu SSH2-Key konvertieren:

```
# ssh-keygen -e -f ~/.ssh/id_dsa.pub > ~/.ssh/id_dsa_ssh2.pub
```

SSH

SSH-Tunnel

Durch SSH-Tunnel ist es möglich, Systeme zu erreichen, die z.B. in einem privaten Netz hinter einer Firewall liegen. Ein typisches Szenario sieht z.B. so aus:

Arbeitsplatz Zuhause 192.168.10.1 ? via NAT-Router ins internet -> firewall@work firewall.beispiel.de ->
Arbeitsplatz in privatem Netz 192.168.1.100

Mit folgendem Befehl kann man den lokalen Port 5900 (für VNC) an die IP 192.168.1.100 Port 5900 auf oder im internen LAN hinter dem Host firewall.beispiel.de leiten. Dadurch wird der Port auf 127.0.0.1:5900 gebunden. Der Parameter -N verhindert das Einloggen auf dem Zielsystem, wenn nur der Tunnel benötigt wird:

```
ssh -N -L 5900:192.168.1.100:5900 root@firewall.beispiel.de
```

Wenn der Tunnel steht und auf dem Arbeitsplatzrechner ein VNC-Server läuft, kann man sich mit einem VNC-Viewer auf 127.0.0.1:5900 verbinden und so den Arbeitsplatzrechner fernsteuern.

Will man den Tunnel auf eine bestimmte IP binden sieht der Aufruf so aus:

```
ssh -N -L <meine IP>:5900:192.168.1.100:5900 root@firewall.beispiel.de
```

GPG / GnuPG / PGP

GPG / GnuPG / PGP Cheat-Sheet

- GPG (bzw. gpg) ? OpenPGP encryption and signing tool
- GnuPG ? GNU Privacy Guard
- PGP ? Pretty Good Privacy

GPG-Tools und Verwaltung unter Linux

Keymanager / Grafische Frontends

- gpa ? GNU Privacy Assistant (für GTK)
- seahorse ? Frontend für GNOME
- kleopatra ? Frontend für KDE
- kgpg ? Frontend ebenfalls für KDE

GPG Basics

neuen Key erstellen

```
$ gpg --gen-key
```

Keys anzeigen

```
# nur öffentliche Schlüssel anzeigen  
$ gpg --list-keys
```

```
# nur private Schlüssel anzeigen  
$ gpg --list-secret-keys
```

Keys exportieren

```
# öffentlichen Schlüssel exportieren:  
$ gpg --export -a "Username/KeyID" > user-pub.asc
```

```
# privaten Schlüssel exportieren:  
$ gpg --export-secret-key -a "Username/KeyID" > user-pub-sec.asc
```

Keys importieren

```
$ gpg --import user-pub.asc  
$ gpg --import user-pub-sec.asc # es wird nicht zw. public/secret keys unterschieden
```

Keys löschen

```
$ gpg --delete-key "Username/KeyID"  
$ gpg --delete-secret-key "Username/KeyID"
```


Fingerprints für den gesamten Schlüsselbund anzeigen

```
$ gpg --fingerprint
```

Trust-Database exportieren/importieren

```
# Export:  
$ gpg --export-ownertrust > trust.txt  
  
# Import:  
gpg --import-ownertrust < trust.txt
```

Widerrufszertifikat erzeugen

Mit einem solchen Zertifikat lassen sich Schlüssel auf Keyservern für ungültig erklären, z.B. wenn man den privaten Schlüssel verloren oder sich einen neuen erstellt hat.

```
$ gpg --gen-revoke "Username/KeyID"
```

Passphrase eines Schlüssels ändern

```
$ gpg --edit-key "Username/KeyID" passwd
```

Vertrauensverhältnis festlegen

```
$ gpg --edit-key "Username/KeyID" trust
```

ICQ und GPG

Die ICQ-UIN als neue UserID an den eigenen Key hängen.

Das ist nicht unbedingt Standard-Konform

1. zuerst die UID rausfinden, sofern nicht bekannt:

```
gpg --list-secret-keys
```

2. mit folgendem Befehl fügt man eine neue (nicht standard-konforme) UID an den Key an:

```
gpg --allow-freeform-uid --edit-key <keyid> adduid
```

3. das Ganze sieht dann etwa so aus:

```
gpg (GnuPG) 1.4.7; Copyright (C) 2006 Free Software Foundation, Inc.  
This program comes with ABSOLUTELY NO WARRANTY.  
This is free software, and you are welcome to redistribute it  
under certain conditions. See the file COPYING for details.  
  
Secret key is available.  
  
[...]
```

Real name: Hans Mustermann

Comment:

You selected this USER-ID:

"Hans Mustermann"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o

You need a passphrase to unlock the secret key for
user: "Hans Mustermann <hans@mustermann.de>"

[...]

Command> quit

Save changes? (y/N) y

PGP public key von Keyserver holen und exportieren

Das folgende kleine Script lädt einen GPG-Pubkey von einem Keyserver herunter und exportiert ihn im ASCII-Format in eine Datei. Als Argument übergibt man ihm die ID des Keys (so wird dann auch die Datei benannt).

Der Key liegt danach ebenfalls im lokalen Keystore.

```
gpg-get-key.sh#!/usr/bin/env bash

#
# download a public key from a keyserver and write it ascii-armored to a file
#

KEYSERVER="keyserver.ubuntu.com"
GPGOPTS="--batch --quiet"

if [[ ${#} -eq 0 ]]; then
    echo "Usage: ${0} <keyid>"
    exit 1
fi

gpg ${GPGOPTS} --keyserver ${KEYSERVER} --recv ${1}

if [[ $? -ne 0 ]]; then
    echo "key '${1}' not found"
    exit 1
fi

gpg ${GPGOPTS} --export --armor -o ${1}.asc ${1}

if [[ -e ${1}.asc ]]; then
    echo "successfully exported the key to '${1}.asc'"
fi
```

TLS / SSL

OpenSSL - TLS-/SSL-Zertifikate

OpenSSL - Zertifikate anzeigen/prüfen/testen

- Zertifikat komplett anzeigen

```
openssl x509 -noout -text -in <zertifikatsname.crt>
```

- den Herausgeber des Zertifikats anzeigen

```
openssl x509 -noout -issuer -in <zertifikatsname.crt>
```

- Für wen wurde das Zertifikat ausgestellt?

```
openssl x509 -noout -subject -in <zertifikatsname.crt>
```

- Für welchen Zeitraum ist das Zertifikat gültig?

```
openssl x509 -noout -dates -in <zertifikatsname.crt>
```

- das obige kombiniert anzeigen

```
openssl x509 -noout -issuer -subject -dates -in <zertifikatsname.crt>
```

- den hash anzeigen

```
openssl x509 -noout -hash -in <zertifikatsname.crt>
```

- den MD5-Fingerprint anzeigen

```
openssl x509 -noout -fingerprint -in <zertifikatsname.crt>
```

- ein SSL-Zertifikat prüfen

```
openssl verify -verbose <zertifikatsname.crt>
```

- einen SSL-Port auf Zertifikate abfragen (Beispiel LDAP)

```
echo QUIT | openssl s_client -connect localhost:636 -showcerts
```

- ein HTTPS-Serverzertifikat runterladen

```
echo QUIT | openssl s_client -connect www.magenbrot.net:443 | sed -ne '/BEGIN CERT/,/END CERT/p'
```

- ein HTTPS-Serverzertifikat runterladen und in lesbar ausgeben

```
echo QUIT | openssl s_client -connect www.magenbrot.net:443 | openssl x509 -noout -text
```

- Gültigkeit eines HTTPS-Serverzertifikats anzeigen

```
echo QUIT | openssl s_client -connect www.magenbrot.net:443 2>/dev/null | openssl x509 -noout -d
```

- bei Webservern mit SNI muss der gewünschte Host im Header mitgeschickt werden, da sonst das SSL-Zertifikat des Default-Vhosts ausgeliefert wird

```
echo QUIT | openssl s_client -tls1_2 -servername ovtec.it -connect ovtec.it:443 | openssl x509 -d
```

- das Gleiche für Mailservices mit STARTTLS (SMTP, IMAP, POP3)

```
# herunterladen und in Datei speichern:
echo QUIT | openssl s_client -starttls smtp -crlf -connect mailgw.ovtec.it:25 | sed -ne '/BEGIN CERTIFICATE/p'
echo QUIT | openssl s_client -starttls pop3 -crlf -connect mail.ovtec.de:110 | sed -ne '/BEGIN CERTIFICATE/p'
echo QUIT | openssl s_client -starttls imap -crlf -connect mail.ovtec.de:143 | sed -ne '/BEGIN CERTIFICATE/p'

# in lesbar ausgeben:
echo QUIT | openssl s_client -starttls smtp -crlf -connect mailgw.ovtec.it:25 | openssl x509 -noout -text
echo QUIT | openssl s_client -starttls pop3 -crlf -connect mail.ovtec.de:110 | openssl x509 -noout -text
echo QUIT | openssl s_client -starttls imap -crlf -connect mail.ovtec.de:143 | openssl x509 -noout -text
```

CSR erzeugen/anzeigen (Certificate Signing Request)

Wenn Ihr ein offizielles Zertifikat bestellen wollt, müsst ihr für die Certificate Authority (CA) einen Zertifikatsantrag erstellen. Dieser wird auf der Webseite Eurer CA (z.B. Thawte, Geotrust, Startssl) hochgeladen und die CA erzeugt daraus dann das endgültige Zertifikat, das ihr dann zusammen mit den Zwischenzertifikaten (intermediate-certificate) herunterladen könnt.

Die aktuellen Browser verlangen die Angabe der DNS-Namen, die im Zertifikat enthalten sein sollen, in einer X509v3 Erweiterung zu OpenSSL als sogenannte „Subject Alternative Names“ (SAN). Der CommonName alleine reicht nicht mehr aus.

Umlaute im Städte- oder Firmennamen sind mit utf8 auch kein Problem mehr.

Ich habe eine Konfigurationsdatei erstellt, die ihr herunterladen und für eure Zwecke anpassen könnt. Das Erstellen eines 4096-Bit Keys und passender CSR ist dann kein Problem mehr und mit einem Kommando erledigt:

```
# Konfigurationsdatei herunterladen und anpassen
wget -O request.cnf https://wiki.magenbrot.net/_media/linux/kryptographie/ssl/request.cnf
vi request.cnf # SAN anpassen!

# 4096-bit Key erstellen und das CSR mit den Angaben aus der Konfigurationsdatei generieren
openssl req -nodes -newkey rsa:4096 -keyout ovtec.it.key -new -out ovtec.it.csr -config request.cnf

# CSR anzeigen lassen (mit UTF8)
openssl req -noout -text -nameopt utf8 -in ovtec.it.csr
```

- ALTE Anleitung: mit diesen Kommandos könnt ihr einen Key und das dazu gehörende CSR erstellen

```
# 4096 Bit RSA-Key erzeugen
openssl genrsa -out <zertifikatsname.key> 4096
```

```
# den CSR dazu erzeugen
openssl req -new -sha256 -key <zertifikatsname.key> -out <zertifikatsname.csr>

#jetzt sind ein paar Fragen zu beantworten (gibt man nur einen . ein so bleibt das Feld leer):
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:Bayern
Locality Name (eg, city) []:Fuerth
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Deine Firma
Organizational Unit Name (eg, section) []:.

Common Name (eg, YOUR name) []:www.meinedomain.de
Email Address []:webmaster@meinedomain.de

Please enter the following 'extra' attributes to be sent with your certificate request
A challenge password []:
An optional company name []:
```

(optional) den Key mit einer Passphrase versehen. Diese Passphrase wird dann z.B. beim Starten von Apache abgefragt. Also Achtung: Ein automatischer Start des Apachen ist dann nur noch mit weiteren Tricks möglich.

```
openssl rsa -des3 -in <zertifikatsname.key> -out <zertifikatsname.key.sec>
```

- einen CSR (Zertifikatsrequest) anzeigen

```
openssl req -noout -text -in <request.csr>
```

Passphrase entfernen/ändern

- Passphrase für ein Keyfile entfernen/ändern (RSA Keys)

```
# Passphrase entfernen
openssl rsa -in <zertifikatsname.key> -out <neueskeyfile.key>

# Passphrase ändern
openssl rsa -des3 -in <zertifikatsname.key> -out <neueskeyfile.key>
```

- Passphrase für ein Keyfile entfernen/ändern (ECC Keys)

```
# Passphrase entfernen
openssl pkey -in <zertifikatsname.key> -out <neueskeyfile.key>

# Passphrase ändern
openssl pkey -des3 -in <zertifikatsname.key> -out <neueskeyfile.key>
```

Selbstsignierte (selfsigned) Zertifikate erstellen

Diese Zertifikate können für interne Zwecke eingesetzt werden oder für den Zeitraum bis man von der Trusted CA sein richtiges Zertifikat bekommt. Mit wenigen Schritten ist ein solches Zertifikat erstellt, diese Beispiel erzeugt ein für 60 Tage gültiges Zertifikat:

```
openssl genrsa -out <zertifikatsname.key> 4096
[...]  
openssl req -new -sha256 -key <zertifikatsname.key> -out <zertifikatsname.csr> #(siehe oben)  
[...]  
openssl x509 -req -sha256 -days 60 -in <zertifikatsname.csr> -signkey <zertifikatsname.key> -out  
[...]
```

oder als komfortabler Einzeiler:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -sha256 -nodes -subj
```

Zertifikate konvertieren

- PEM nach DER

```
openssl x509 -outform der -in certificate.pem -out certificate.der
```

- PEM nach P7B

```
openssl crl2pkcs7 -nocrl -certfile certificate.cer -out certificate.p7b -certfile CACert.cer
```

- PEM nach PKCS12 (P12)

```
openssl pkcs12 -export -out certificate.p12 -inkey userkey.pem -in usercert.pem
```

- PEM nach PFX

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

- DER nach PEM

```
openssl x509 -inform der -in certificate.cer -out certificate.pem
```

- P7B / PKCS7 nach PEM

```
openssl pkcs7 -print_certs -in certificate.p7b -out certificate.cer
```

- P7B / PKCS7 nach PFX

```
openssl pkcs7 -print_certs -in certificate.p7b -out certificate.cer  
openssl pkcs12 -export -in certificate.cer -inkey privateKey.key -out certificate.pfx -certfile
```

- PFX / PKCS12 nach PEM

```
# es wird nach dem Import-Passwort gefragt, wenn eines gesetzt wurde:  
openssl pkcs12 -in certificate.pfx -out certificate.pem -nodes  
  
# beim Fehler "Mac verify error: invalid password?" bitte mit der nächsten Methode probieren
```

- PFX / PKCS12 nach PEM (wenn er das Passwort nicht frisst):

```
openssl pkcs12 -in certificate.pfx -out certificate.pem -nodes -password pass:<PASSWORT>
```


PEM-Datei richtig erstellen

Seit Apache 2.4.8 ist SSLCertificateChainFile als deprecated markiert. Das heißt, dass nun Zertifikat und Zwischenzertifikat in einer gemeinsamen Datei gespeichert werden müssen und über die Direktive SSLCertificateFile geladen werden.

Dabei ist die Reihenfolge wichtig. Eine solche Datei wird folgendermaßen erstellt:

```
cat meine-domain.de.crt ssl-ca-intermediate.crt > meine-domain.de-bundle.crt
```

Ganz selten wird auch das Root der SSL-CA benötigt, das hängt dann einfach ganz am Ende dran:

```
cat meine-domain.de.crt ssl-ca-intermediate.crt ssl-ca-root.crt > meine-domain.de-bundle.crt
```

Um solche Dateien einfach zu erkennen, füge ich ein -bundle in den Dateinamen ein.

PEM inklusive private Key

Es kann auch noch der SSL-Key eingefügt werden. Dieser kommt dann an erster oder letzter Stelle. Die Zertifikate übrigen in der üblichen Reihenfolge (Zertifikat, Intermediate und gegebenenfalls noch das Root-Zertifikat). Apache unterstützt das zwar, es wird allerdings aktuell nicht empfohlen.

```
cat meine-domain.de.key meine-domain.de.crt ssl-ca-intermediate.crt (ssl-ca-root.crt) > meine-domain.de-bundle.crt
```

Folgender Einzeiler baut das entsprechende PEM-File zusammen, falls Key und Zertifikat in zwei Dateien mit den Endungen .key und .crt vorliegen:

```
for i in `ls *.key`; do NAME=$(echo $i | sed 's/\.key//'); echo $NAME; cat $NAME.crt $NAME.key >> $NAME-bundle.crt
```

Prüfen ob ein Zertifikat zu einem Key passt

Der private Teil eines Schlüssels enthält verschiedene Zahlen. Zwei dieser Zahlen bilden den „Public Key“ (diese Zahlen sind dann auch im Zertifikat erhalten), der Rest gehört zum „Private Key“. Um zu prüfen, ob der public key zum private key passt, können diese beiden Zahlen ausgelesen und verglichen werden.

```
# Zertifikate
$ openssl x509 -noout -modulus -in server.crt | openssl md5
# Private Schlüssel
$ openssl rsa -noout -modulus -in server.key | openssl md5
# Zertifikatsanforderung
$ openssl req -noout -modulus -in server.csr | openssl md5
```

oder als einfach zu verwendendes Script:

```
#!/bin/bash
#
# check if certificate, signing request and key match
#
# $Id: check-certificate.sh 524 2016-01-15 11:30:11Z magenbrot $
# zlfamous added intermediate and key size check
```

```

#
if [ "x$1" = "x" ]; then
    echo "Usage: $0 <filename without .key, .crt, .csr or .intermediates>"
    exit 1
fi

if [ -e $1.key ]; then
    output="$1.key: `openssl rsa -noout -modulus -in $1.key | openssl md5 | cut -d" " -f2`"
    key_size=`openssl rsa -noout -text -in $1.key | grep "Private-Key" | cut -d" " -f2 | cut -d "("`
    if [ $key_size -lt 4096 ]; then
        output="$output \e[39m(key size: \e[33m$key_size\e[39m bit)"
    else
        output="$output \e[39m(key size: \e[32m$key_size\e[39m bit)"
    fi
    echo -e $output
else
    echo "$1.key: file not found"
fi

if [ -e $1.csr ]; then
    echo -n "$1.csr: "
    openssl req -noout -modulus -in $1.csr | openssl md5 | cut -d" " -f2
else
    echo "$1.csr: file not found"
fi

if [ -e $1.crt ]; then
    echo -n "$1.crt: "
    openssl x509 -noout -modulus -in $1.crt | openssl md5 | cut -d" " -f2
else
    echo "$1.crt: file not found"
fi

if [ -e $1.intermediates ]; then
    echo -n "$1.intermediates: "
    subject=`openssl x509 -noout -subject_hash -in $1.intermediates`
    issuer=`openssl x509 -noout -issuer_hash -in $1.crt`
    if [ "$subject" != "" -o "$issuer" != "" ]; then
        if [ "$subject" == "$issuer" ]; then
            signature=`openssl x509 -noout -text -in $1.intermediates | grep "Signature Algorithm:" |`
            echo -e "\e[32missuer matches subject \e[39m- signature hash: \e[32m$signature\e[39m"
        else
            echo -e "\e[31missuer doesn't match subject"
        fi
    fi
else
    echo "$1.intermediates: file not found"
fi

```

Diffie-Hellman Parameterdatei Bitgröße ermitteln

Auf Diffie-Hellman aufsetzende Cipher benötigen eine entsprechende Parameterdatei. Die Standardgröße beträgt 1024 bit. Die Empfehlung nach der Logjam-Attacke sind mind. 2048 bit, besser 4096 bit.

Die Parameterdatei wird folgendermaßen erstellt:

```
openssl dhparam -out dhparams.pem 4096
```

Aus der Datei läßt sich die Bitgröße nicht auf Anhieb herauslesen. Mit diesem Befehl lässt sich die Info darstellen:

```
openssl dhparam -inform PEM -in dhparams.pem -check -text
```

(Zwischen-)Zertifikat Key und Zertifikatsrequest per Script auf Plausibilität prüfen

Dieses Script prüft folgende Punkte:

- mind. 4096 Bit Key
- Passen CSR, CRT, KEY zusammen (openssl modulus / md5)?
- Signatur-Hash prüfen
- Passt das Intermediate-CRT zum CRT

Namenskonvention der Dateien:

- meine-domain.de.key
- meine-domain.de.crt
- meine-domain.de.csr
- meine-domain.de.intermediates

```
#!/bin/bash

#
# check if certificate, signing request and key match
#

if [ "$1" = "" ]; then
    echo "Usage: $0 <filename without .key, .crt, .csr or .intermediates>"
    exit 1
fi

if [ -e $1.key ]; then
    output="$1.key: `openssl rsa -noout -modulus -in $1.key | openssl md5 | cut -d" " -f2`"
    key_size=`openssl rsa -noout -text -in $1.key | grep "Private-Key" | cut -d" " -f2 | cut -d "(" -f2`
    if [ $key_size -lt 4096 ]; then
        output="$output \e[39m(key size: \e[33m$key_size\e[39m bit)"
    else
        output="$output \e[39m(key size: \e[32m$key_size\e[39m bit)"
    fi
    echo -e $output
else
```

```

    echo "$1.key: file not found"
fi

if [ -e $1.csr ]; then
    echo -n "$1.csr: "
    openssl req -noout -modulus -in $1.csr | openssl md5 | cut -d" " -f2
else
    echo "$1.csr: file not found"
fi

if [ -e $1.crt ]; then
    echo -n "$1.crt: "
    openssl x509 -noout -modulus -in $1.crt | openssl md5 | cut -d" " -f2
else
    echo "$1.crt: file not found"
fi

if [ -e $1.intermediates ]; then
    echo -n "$1.intermediates: "
    subject=`openssl x509 -noout -subject_hash -in $1.intermediates`
    issuer=`openssl x509 -noout -issuer_hash -in $1.crt`
    if [ "$subject" != "" -o "$issuer" != "" ]; then
        if [ "$subject" == "$issuer" ]; then
            signature=`openssl x509 -noout -text -in $1.intermediates | grep "Signature Algorithm:" | cut -d" " -f7 | head
-n1`
            echo -e "\e[32missuer matches subject \e[39m- signature hash: \e[32m$signature\e[39m"
        else
            echo -e "\e[31missuer doesn't match subject"
        fi
    fi
    chown root:root $1.intermediates
    chmod 0600 $1.key $1.csr $1.crt $1.intermediates
else
    echo "$1.intermediates: file not found"
fi

```

CAcert Root Zertifikat unter Debian/Ubuntu einbinden

Da das CAcert Root Zertifikat leider nicht mehr mit Debian und damit auch nicht mit Ubuntu ausgeliefert wird folgt hier eine Anleitung, wie man es im System einbindet. Das betrifft nur das Betriebssystem, ggf. muss das Zertifikat auch in euren Browser importiert werden.

Dazu müssen die CAcert Root Zertifikate (root.crt und class3.crt) in einem Verzeichnis unterhalb von /usr/local/share/ca-certificates abgelegt werden und das Zertifikatsbundle neu erzeugt werden.

```
mkdir /usr/local/share/ca-certificates/cacert.org
wget -P /usr/local/share/ca-certificates/cacert.org http://www.cacert.org/certs/root.crt http://
update-ca-certificates
```

Signatur-Algorithmen einer Zertifikatskette anzeigen

Da mit SHA1 signierte Zertifikate inzwischen als unsicher eingestuft werden, habe ich nach einer einfachen Möglichkeit gesucht, wie ich herausfinden kann, welche meiner Zertifikate davon betroffen sind.

Es reicht allerdings nicht, nur das Serverzertifikat auszutauschen. Es sollte auch die Zertifikatskette (Certificate chain) untersucht werden, da ggf. auch das Zwischen- und CA-Zertifikat ausgetauscht werden muss. Wobei das CA-Zertifikat natürlich nur von der CA, bzw. dem Herausgeber selbst (Geotrust, Thawte, etc) ausgetauscht werden kann.

Aufgerufen wird das Script so (kann dann z.B. in einer Schleife mit euren verschiedenen Servern/Ports gefüttert werden):

```
# ./check-ssl-chain.sh www.heise.de:443
Signature Algorithm: sha256WithRSAEncryption
Subject: C=DE, ST=Niedersachsen, L=Hannover, O=Heise Zeitschriften Verlag GmbH und Co KG
Signature Algorithm: sha256WithRSAEncryption

Signature Algorithm: sha256WithRSAEncryption
Subject: C=US, O=thawte, Inc., CN=thawte SHA256 SSL CA
Signature Algorithm: sha256WithRSAEncryption
```

oder:

```
# ./check-ssl-chain.sh google.com:443
Signature Algorithm: sha1WithRSAEncryption
Subject: C=US, ST=California, L=Mountain View, O=Google Inc, CN=google.com
Signature Algorithm: sha1WithRSAEncryption

Signature Algorithm: sha1WithRSAEncryption
Subject: C=US, O=Google Inc, CN=Google Internet Authority G2
Signature Algorithm: sha1WithRSAEncryption

Signature Algorithm: sha1WithRSAEncryption
Subject: C=US, O=GeoTrust Inc., CN=GeoTrust Global CA
Signature Algorithm: sha1WithRSAEncryption
```

Das Beispiel 1 (Heise) zeigt, dass der Admin fleissig war und die Zertifikate schon gegen SHA256-signierte ausgetauscht hat. Google hingegen setzt noch mit SHA1 signierte Zertifikate ein.

Hier noch das Script:

```
#!/bin/sh

HOST=$1
TMP=`mktemp -d`

echo QUIT | openssl s_client -showcerts -connect $HOST 2>/dev/null | sed -ne
'/BEGIN CERT/,/END CERT/p' | awk -v TMP="$TMP" '/BEGIN/{n++;}{print >TMP"/out" n ".crt" }'

for i in `ls $TMP/out*`; do
    openssl x509 -in $i -noout -text | egrep "Signature Algorithm|Subject:"
    echo
done

rm -rf $TMP
```


VPN (Wireguard / OpenVPN / IPSec)

OpenVPN nach Hause

Serverseite

hier bei mir dient die Fritz!Box 7050 als OpenVPN Server(via Firmware-Mod von <http://www.freetz.org>). Die Konfiguration sieht so aus:

```
#
dev tun0
dev-node /dev/misc/net/tun
ifconfig 192.168.200.2 192.168.200.1
tun-mtu 1500
float
mssfix

#Pfad zum Key File
secret /var/tmp/secret.key

#Protokoll auf TCP und Port 1199
proto tcp-server
port 1199

#Protokollierung auf 4
verb 4

#daemon

#Routen setzen, bei route Subnetz des Clients eintragen
route 192.168.150.0 255.255.255.0

#Verbindung erhalten
ping 15
ping-restart 120
```

Clientseite

als Client dient mein PC im Büro, das hat den Vorteil, das man in der Bürofirewall kein Loch aufreissen muss.

```
ifconfig 192.168.200.1 192.168.200.2
dev tun
#dev-node /dev/misc/net/tun
tun-mtu 1500
mssfix
persist-tun
persist-key

#Remote Adresse des Servers angeben
#muss entsprechend geaendert werden
remote remoteserver.net

#Pfad zum Key File
secret /etc/openvpn/secret.key

#Protokoll auf TCP und Port 1199
proto tcp-client
port 1199

#Da die Verbindung alle 24 Stunden getrennt wird
#soll regelmässig kontrolliert werden ob die Verbindung noch steht
ping 15
ping-restart 120

#Der DynDNS-Name soll alle 60 Sekunden neu aufgelöst werden
#da OpenVPN sonst ständig versucht die alte IP
```

```
#zu verbinden
resolv-retry 60

#Protokollierungseinstellung
#4 ist optimaler Modus
verb 4

#Daemon sollte erst eingeschaltet werden wenn die
#Konfiguration passt
daemon

#Routen setzen, bei route Subnetz der Server-Box eintragen
route 192.168.10.0 255.255.255.0
push "route 192.168.150.0 255.255.255.0"
```

für die Konfiguration der Fritz!Box bin ich nach dieser Anleitung vorgegangen:

<http://www.tecchannel.de/server/linux/435560/>

Wenn alles klappt ist das Netz zuhause mit dem internen Netz des Clients (und umgekehrt) verbunden. Ist hinter dem Client kein Netz vorhanden sollte man vorher z.B. eine zusätzliche interne IP vergeben (z.B. 192.168.150.1).

IPsec Roadwarrior-VPN via racoon

Dieses Dokument beschreibt die Konfiguration und den Betrieb eines VPN mit Preshared-Keys und Racoon unter Fedora Core 4.

- Falls nicht vorhanden, das Verzeichnis „/etc/racoon“ anlegen, hier werden alle Configfiles abgelegt

/etc/racoon/setkey.conf

```
#!/sbin/setkey -f

# Flush the SAD and SPD
flush;
spdflush;

#####
# Roadwarrior <-> Gateway

# 123.123.123.123 = externe IP des Gateways
# 192.168.1.0/24 = internes Netz auf Gateway-Seite

# HOST to HOST
spdadd 123.123.123.123 0.0.0.0 any -P out ipsec
        esp/tunnel/123.123.123.123-0.0.0.0/require;
spdadd 0.0.0.0 123.123.123.123 any -P in ipsec
        esp/tunnel/0.0.0.0-123.123.123.123/require;

# HOST to LAN
spdadd 192.168.1.0/24 0.0.0.0 any -P out ipsec
        esp/tunnel/123.123.123.123-0.0.0.0/require;
spdadd 0.0.0.0 192.168.1.0/24 any -P in ipsec
        esp/tunnel/0.0.0.0-123.123.123.123/require;
#####
```

- mit „chmod 0700 /etc/racoon/setkey.conf“ lesen/schreiben/ausführen für root setzen.

/etc/racoon/racoon.conf

```
path include "/etc/racoon";
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";

# Preshared Keys
remote anonymous {
    exchange_mode aggressive, main, base;
    #doi ipsec_doi;
    nat_traversal on;
    generate_policy on;
    passive on;
    #my_identifier address 212.34.164.18;
    peers_identifier user_fqdn;
    proposal {
        encryption_algorithm 3des;
        hash_algorithm md5;
        authentication_method pre_shared_key;
        dh_group modp1024;
    }
}

sainfo anonymous {
    pfs_group modp1024;
    encryption_algorithm 3des;
```

```
        authentication_algorithm hmac_md5;
        compression_algorithm deflate;
    }
```

- mit „chmod 0600 /etc/racoon/racoon.conf“ lesen/schreiben für root setzen.

/etc/racoon/psk.txt enthält die PresharedKeys in folgendem Format:

```
roadwarrior001@gateway.de MpfeEuwPEkov7ScUtKtmAa4FGWVda9jjtruesrkJKUx8sWC4u9
```

- mit „chmod 0600 /etc/racoon/psk.txt“ lesen/schreiben für root setzen.

/etc/sysconfig/racoon

```
OPTS="-f /etc/racoon/racoon.conf -l /var/log/racoon -v"
```

- mit „chmod 0644 /etc/sysconfig/racoon“ die Berechtigungen setzen

/etc/init.d/racoon

```
#!/bin/bash
#
# racoon          Start/Stop the racoon IKE daemon.
#
# chkconfig: 2345 90 60
# description: racoon is the IKE daemon of the KAME tools. Use it with \
#              the native Linux 2.6 IPsec-Stack

# processname: racoon
# config: /etc/racoon/racoon.conf
# pidfile: /var/run/racoon.pid

# Source function library.
. /etc/init.d/functions

OPTS=""

[ -f /etc/sysconfig/racoon ] && . /etc/sysconfig/racoon

RETVAL=0

prog="racoon"

start() {
    /etc/racoon/setkey.conf
    echo -n $"Starting $prog: "
    daemon racoon $OPTS
    RETVAL=$?
    echo
    [ $RETVAL -eq 0 ] && touch /var/lock/subsys/racoon
    return $RETVAL
}

stop() {
    echo -n $"Stopping $prog: "
    killproc racoon
    RETVAL=$?
    echo
    [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/racoon
    return $RETVAL
}

rhstatus () {
    status racoon
}

restart () {
    stop
```

```

    start
}

reload () {
    echo -n $"Reloading racoon daemon configuration: "
    killproc racoon -HUP
    RETVAL=$?
    echo
    return $RETVAL
}

case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    restart)
        restart
        ;;
    reload)
        reload
        ;;
    status)
        rhstatus
        ;;
    condrestart)
        [ -f /var/lock/subsys/crond ] && restart || :
        ;;
    *)
        echo $"Usage: $0 {start|stop|status|reload|restart|condrestart}"
        exit 1
esac

exit $?

```

- mit „chmod 0744 /etc/init.d/racoon“ die Berechtigungen setzen.
- ein „chkconfig –add racoon“ aktiviert das Script beim Booten
- mit „service racoon start“ die Security Policy Database (SPD) laden (setkey.conf) und den Racoon-Dämon starten
- geloggt wird nach /var/log/racoon
- Um den Debuglevel zu erhöhen ggf. in /etc/sysconfig/racoon die -v Option um weitere v ergänzen, z.B.

```
OPTS="-f /etc/racoon/racoon.conf -l /var/log/racoon -vvv"
```

Tunnelüberwachung via Script

Mit dem folgenden Script lässt sich bequem ein Tunnel überwachen und ggf. automatisch neu starten:

```
#!/bin/bash
# keepalive for ipsec
# quick'n'dirty hack

# Check darf 3x fehlschlagen, dann wird der Tunnel neu gestartet
failmax=3

# Check alle 10 Sekunden durchführen
keepalive=10

# eine interne IP der Gegenseite eintragen
CHECKIP="192.168.10.10"

# hier den Tunnelnamen eintragen
CHECKNAME="Aussenstelle1"

# don not edit anything beyond this point!
#####

fail=0

MESSAGE=""

while (true); do

# Achtung: der Ping-Befehl hat nicht in allen Versionen die -I Option (von welchem Device/IP aus

# als -I eth0 das Interface mit der internen IP eintragen
if ping -w 2 -c 1 -s 1 -I eth0 $CHECKIP 2>&1 > /dev/null;
then
    MESSAGE="Tunnel $CHECKNAME OK (check $RANDOM)"
    logger -p local2.info -t TUNNEL "$MESSAGE"
    fail=0
else
    fail=`echo $fail+1|bc`
    MESSAGE="Tunnel $CHECKNAME DOWN: $fail (check $RANDOM)"
    logger -p local2.info -t TUNNEL "$MESSAGE"
fi

if [ $fail -gt $failmax ] ;
then
    MESSAGE="Maxfail ($failmax) reached: restarting Tunnel $CHECKNAME (check
$RANDOM)"
    logger -p local2.info -t TUNNEL "$MESSAGE"

    # Fehler, Tunnel stoppen:
    # wenn als Software Racoon zum Einsatz kommt:
    /etc/init.d/racoon stop

# das hier bei OpenSwan einkommentieren und obiges raus (CHECKNAME muss mit dem Tunnelnamen in d

    #ipsec auto --down $CHECKNAME
    sleep 5

    # jetzt den Tunnel wieder starten:
    # wenn als Software Racoon zum Einsatz kommt:
    /etc/init.d/racoon start

# das hier bei OpenSwan einkommentieren und obiges raus (CHECKNAME muss mit dem Tunnelnamen in d

    #ipsec auto --up $CHECKNAME
    sleep 120
    fail=0
fi
sleep $keepalive
done
```

das (check \$RANDOM) ist als Workaround für Syslog gedacht, dort würden sonst massig „last message repeated xx times“ auftauchen.

das Script loggt in die Syslog-Facility local2. Folgender Eintrag ist für syslog vorzunehmen, um nach /var/log/tunnel.log zu loggen:

```
local2.*                                /var/log/tunnel.log
```

dieser Eintrag ist für syslog-NG

```
source s_sys {
    file ("/proc/kmsg" log_prefix("kernel: "));
    unix-stream("/dev/log");
    udp(ip(0.0.0.0) port(514));
    internal();
};

# /var/log/tunnel.log
filter f_tunnel { facility(local2); };
destination d_tunnel { file("/var/log/tunnel.log"); };
log { source(s_sys); filter(f_tunnel); destination(d_tunnel); };
```

es kann natürlich auch jede andere Facility verwendet werden.

um das Script bei Reboot automatisch wieder zu starten folgenden Eintrag in /etc/rc.local vornehmen:

```
/usr/local/sbin/probe.ipsec 2>&1 >> /dev/null &
```

Das Init-Script für Racoon ist hier zu finden: [Roadwarrior-VPN via racoon](#)

Das Script ist für Racoon in dieser Form leider nicht optimal, da alle bestehenden Tunnel gekillt werden. Das Script ist nur sinnvoll für ein Gateway mit nur einem Tunnel.

OpenVPN startet nicht (ca md too weak)

Die folgende Lösung sollte nur ein kurzfristiger Workaround bleiben. Sicherer wäre es wenn auch der Serverteil aktualisiert und auf aktuelle Hashes und Cipher umgestellt wird!

Problem: OpenVPN Tunnel mag nicht starten. Im Log kommt folgende Meldung:

```
(OpenSSL: error:140AB18E:SSL routines:SSL_CTX_use_certificate:ca md too weak)
```

Neue OpenVPN Versionen haben veraltete Hashes und Ciphers deaktiviert. Es laufen aber noch ältere OpenVPN-Server zu denen man sich jetzt nicht mehr verbinden kann.

Die älteren Ciphers lassen sich mit einem Konfigurationsschalter wieder aktivieren.

Bei OpenVPN Configs direkt diese Zeile eintragen:

```
tls-cipher "DEFAULT:@SECLEVEL=0"
```

Bei Verbindungen mit dem NetworkManager kommt diese Zeile in den [vpn]-Teil:

```
tls-cipher=DEFAULT:@SECLEVEL=0
```

Danach muss der NetworkManager neu gestartet werden (Achtung, alle Verbindungen werden unterbrochen).
Achtung: Die Zeile verschwindet auch wieder aus der Config wenn in der GUI Änderungen vorgenommen wurden.