SSH

SSH-Keys erzeugen und verwalten

SSH-Key erzeugen

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

Es wird ein 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 Zielserver 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_authtok
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 angemeckerte 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 ;)

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

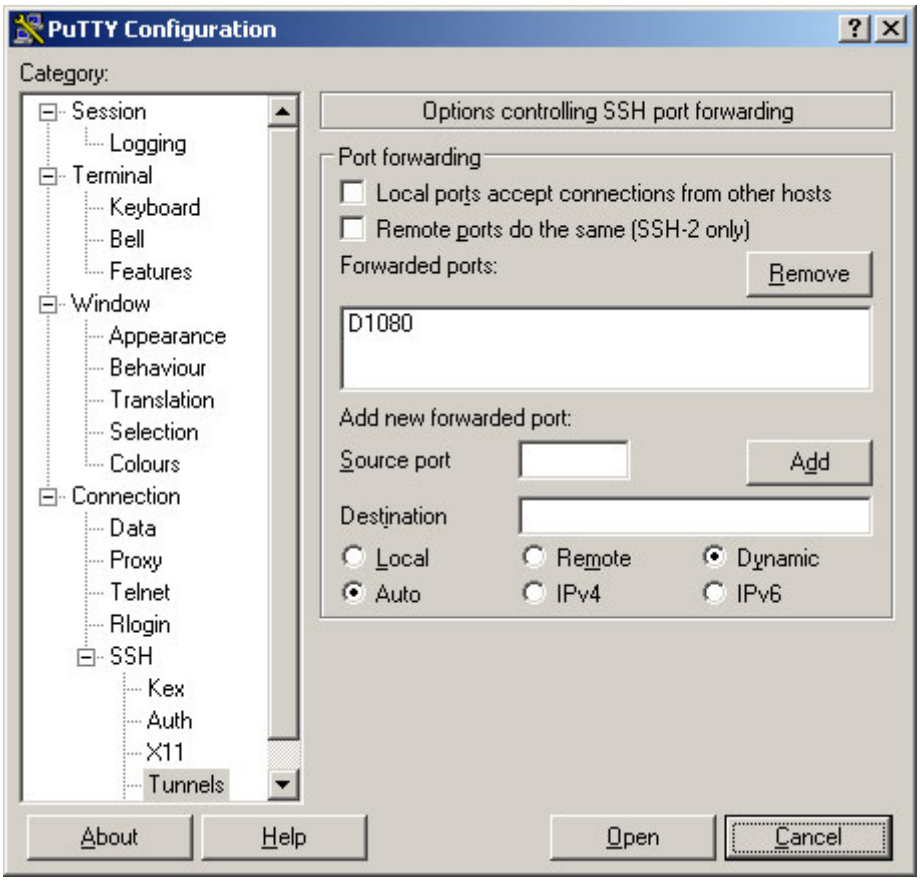
Sinn eines SocksProxy ist es z.B. auf Server und Dienste innerhalb eines Firmennetzwerks zugreifen zu können ohne jedoch dafür Ports in der Firewall aufreissen zu müssen oder auch um Netzwerkverkehr verschlüsselt über einen anderen PC zu tunneln.

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-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-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
```