

Mar 27, 18 7:53

ssh-HOWTO.txt

Page 1/15

HOWTO zu SSH (Secure Shell)

(C) 2007-2017 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>  
 OSTC Open Source Training and Consulting GmbH  
<http://www.ostc.de>

\$Id: ssh-HOWTO.txt,v 1.59 2018/03/27 05:52:53 tsbirn Exp \$

Dieses Dokument beschreibt Grundlagen und Einsatz von SSH (Secure Shell)  
 unter UNIX/Linux (sicheres telnet, rcp, rsh, rlogin, ftp + "VPN für Arme").

---

## INHALTSVERZEICHNIS

- 1) Einführung
- 2) Übersicht Konfigurations- und Schlüsseldateien
- 3) Prinzipielle Arten von Verschlüsselung
- 4) SSH installieren, aktivieren und testen
  - 4.1) SSH-Daemon "sshd"
  - 4.2) SSH-Protokoll
  - 4.3) SSH-Client "ssh"
- 5) Einsatz von SSH
  - 5.1) Anmelden per SSH-Client
  - 5.2) Server-Authentifizierung
  - 5.3) Kommando per SSH remote ausführen
  - 5.4) Dateien per SSH zwischen Rechnern kopieren
  - 5.5) X Window-Protokoll tunneln
  - 5.6) Schlüsselbasierte Client-Authentifizierung einrichten (kein Passwort)
    - 5.6.1) Grafik zum Ablauf einer Client-Authentifizierung
  - 5.7) Privaten Schlüssel mit Passphrase sichern
  - 5.8) SSH-Agent zum Schlüssel verwalten
  - 5.9) Bei SSH-Anmeldung Kommando ausführen/Optionen setzen
- 6) Port-Forwarding (Tunneling von Protokollen)
- 7) Übersicht
  - 7.1) Kommandos
  - 7.2) Server-Konfigurations-Dateien
  - 7.3) Client-Konfigurations-Dateien
  - 7.4) Login-Shell Konfigurations-Dateien
- 8) Tipps
- 9) Links

---

## 1) Einführung

SSH (Secure Shell) stellt mit "ssh", "scp" und "sftp" (Clients) + "sshd" (Server) eine gesicherte (verschlüsselte) Kommunikation über unsichere Netze zur Verfügung, und bildet einen sicheren Ersatz für die Standard-UNIX-Tools "telnet", "ftp", "rlogin", "rcp", "rsh", "rcmd", ... (Remote Login, Remote Kommando-Ausführung bzw. Dateitransfer) mit folgenden Eigenschaften:

- \* Leistungsfähige Authentifizierung von Client UND Server
- \* Sichere Verschlüsselung der übertragenen Daten (und Anmeldung)
- \* Integritätssicherung (Verifizierung) der übertragenen Daten (verhindert "Man-in-the-middle" Attacken)

SSH sollte daher aus Sicherheitsgründen alle oben aufgezählten veralteten UNIX/LINUX-Remote-Kommandos ersetzen. Bei den meisten LINUX/UNIX-Versionen ist SSH inzwischen Standard, auf ftp/telnet wird von vornherein verzichtet:

Weiterhin unterstützt SSH den Schutz von X11-Verbindungen (X Window System) und die Weiterleitung ("Port-Forwarding") von beliebigen TCP-Verbindungen über einen kryptografisch gesicherten Kanal und bietet somit VPN-Funktionalität ("VPN für Arme"). Kurzer Funktionsumfang:

- \* Login auf Remote-Host
- \* Ausführen von Kommandos auf Remote-Host (interaktiv / nicht-interaktiv)
- \* Kopieren von Dateien zwischen Hosts
- \* Port-Forwarding/Tunneling (VPN)
- \* Komprimierung der Daten während Transfer

Eigenschaften:

- \* Automatische und transparente Verschlüsselung der gesamten Kommunikation
- \* Sechs Verfahren der Client-Authentifizierung gegenüber dem Server möglich
- \* RSA-basierte Authentifizierung (IP-, Routing-, DNS-Spoofing nicht möglich)
- \* Systemweite + benutzerbezogene Konfigurations-Dateien
- \* Übertragen von Binärdaten (+ Komprimierung) möglich
- \* Arbeitet gut mit anderen Tools zusammen (z.B. "rsync", "X", "cvs", "svn", "unison").
- \* Port 22 (SSH-Port) für Kommunikation verwendet

Bekannte Implementierungen:

- \* OpenSSH: Kommt ohne patentierte Algorithmen aus (--> <http://www.openssh.org>)
- \* Windows-Client: Putty (suchen in Google nach: putty download)

## 2) Übersicht Konfigurations- und Schlüsseldateien

Es gibt zentrale (rechnerspezifische) Konfigurations-Dateien im Verz. "/etc/ssh" und benutzerspezifische im Verz. "~/.ssh" jedes Benutzers. Wird eine Einstellung in beiden Konfigurations-Dateien getroffen, dann überschreibt die benutzerspezifische Einstellung die zentrale. Existiert die benutzerspezifische Konfigurations-Datei nicht, dann gilt die zentrale vollständig.

Vorrang SSH-Client-Konfiguration (von oben nach unten absteigend):

-o OPT=VAL ...	Kommandozeilen-Option (mehrfach angebbar)
-F FILE	Konf.datei (zentrale + benutzerspez. ignoriert)
~/.ssh/config	Benutzerspezifische Konfiguration
/etc/ssh/ssh_config	Zentrale Konfiguration

Vorrang SSH-Server-Konfiguration (von oben nach unten absteigend):

-o OPT=VAL ...	Kommandozeilen-Option (mehrfach angebbar)
-f FILE	Zentrale Konf.datei explizit angeben
/etc/ssh/sshd_config	Zentrale Konfiguration (Std falls -f FILE fehlt)

Zur Verschlüsselung der Kommunikation gibt es mind. ein Schlüsselpaar pro Rechner und mind. eines pro Benutzer. Da es mehrere Verschlüsselungsverfahren gibt (RSA/DSA/ECDSA/ED25519 --> veraltet/modern, unsicher/sicher, patentbehaltet/patentfrei), können auch bis mehrere Schlüsselpaare existieren, von denen pro Authentifizierungsvorgang eines ausgewählt wird.

- \* Schlüsselpaare für Rechner werden bei der Installation des Rechners bzw. des SSH-Paketes AUTOMATISCH angelegt (per "ssh-keygen").
- \* Schlüsselpaare für Benutzer müssen MANUELL pro Benutzer per Kommando "ssh-keygen" angelegt werden.

Ein zusammengehörendes Schlüsselpaar wird in 2 Dateien abgelegt:

```
XXX      # Privater Teil (bleibt auf Rechner, nur für Eigentümer lesbar)
XXX.pub  # Öffentlicher Teil ("pub"=public, darf beliebig verteilt werden)
```

Konfigurations-Dateien der beiden Kommunikations-Partner:

Lokal (Client)		Remote (Server)
Client: ssh, scp, sftp Putty, ...	<=====> Kommunikation	Server-Daemon: sshd Firewall mit offenem SSH-Port 22
/etc/ssh/ssh_config ~/.ssh/config (ACHTUNG: kein d=daemon)	Konfiguration	/etc/ssh/moduli /etc/ssh/sshd_config (ACHTUNG: d=daemon)
/etc/ssh/ssh_known_hosts ~/.ssh/known_hosts	<=====> Server-Authentifiz.	/etc/ssh/ssh_host_rsa_key (P2) /etc/ssh/ssh_host_rsa_key.pub /etc/ssh/ssh_host_dsa_key /etc/ssh/ssh_host_dsa_key.pub /etc/ssh/ssh_host_ecdsa_key /etc/ssh/ssh_host_ecdsa_key.pub /etc/ssh/ssh_host_ed25519_key /etc/ssh/ssh_host_ed25519_key.pub /etc/ssh/ssh_host_key (P1) /etc/ssh/ssh_host_key.pub
~/.ssh/id_rsa (P2) ~/.ssh/id_rsa.pub ~/.ssh/id_dsa ~/.ssh/id_dsa.pub ~/.ssh/id_ecdsa ~/.ssh/id_ecdsa.pub ~/.ssh/id_ed25519 ~/.ssh/id_ed25519.pub ~/.ssh/identity (P1) ~/.ssh/identity.pub	=====> Client-Authentifiz.	~/.ssh/authorized_keys

Mar 27, 18 7:53

## ssh-HOWTO.txt

Page 3/15

	=====>	/etc/ssh/sshrd
	Login-Shell	~/.ssh/rc
	Konfiguration	~/.ssh/environment
+-----+	+-----+	+-----+
	=====>	~/.hushlogin
	Login-	/etc/motd
	Verhalten	/etc/nologin
+-----+	+-----+	+-----+
	=====>	~/.rhosts
	Host-based	~/.shosts
	Login	/etc/hosts.equiv
		/etc/shosts.equiv
+-----+	+-----+	+-----+
	=====>	/etc/hosts.allow
	TCP-Wrapper	/etc/hosts.deny
+-----+	+-----+	+-----+

## HINWEISE:

- \* Server-Konfigurations-Datei "sshd\_config" (MIT d=daemon!) und Client-Konfigurations-Datei "ssh\_config" (OHNE d=daemon!) NICHT verwechseln!
- \* Besitzverhältnisse und Zugriffsrechte sämtlicher Konfigurations-Dateien und ihrer Verz.pfade müssen einige Bedingungen erfüllen, damit SSH eine schlüsselbasierte Anmeldung erlaubt --> 8) Tipps

## 3) Prinzipielle Arten von Verschlüsselung

- a) SYMMETRISCH: Gleicher Schlüssel zum Ver- und zum Entschlüsseln:
  - Schlüsselaustausch schwierig (muss über geheimen Kanal erfolgen)
  - Schlüssel muss geheim gehalten werden
  - + Schlüssel kurz (128-256 Bit)
  - + Schnell (Schlüsselgenerierung und Verschlüsselung)
- b) ASYMMETRISCH: Ein Schlüsselpaar bestehend aus einem öffentlichen (public key) und einem privaten (private key) Teil:
  - + Schlüsselaustausch einfach (den public key darf JEDER kennen!)
  - + Nur privater Schlüssel muss geheim gehalten werden
  - Schlüssel lang (1024-16384 Bit)
  - Langsam (Schlüsselgenerierung und Verschlüsselung)
- c) HYBRID: Verbindung der Vorteile beider Verfahren.  
Beim Verbindungsaufbau wird über eine asymmetrische Verschlüsselung ein symmetrischer "Sitzungsschlüssel" vereinbart und darüber die eigentliche Kommunikation durchgeführt.

SSH verwendet hybride Verschlüsselung: Das asymmetrische Verfahren RSA (Rivest, Shamir, Adelman), DSA (Digital Signature Algorithm), ECDSA (Elliptic Curve Digital Signature Algorithm) oder ED25519 (Elliptic D?) wird zu Authentifizierung der Gegenstellen verwendet. Danach tauscht SSH einen symmetrischen Schlüssel aus, der die Sitzungsdaten verschlüsselt. Dieser Schlüssel wird in bestimmten Zeitabständen (etwa alle 60 Minuten) erneuert.

Grund für den ständigen Wechsel des Sitzungsschlüssels: Eine aufgezeichnete SSH-Sitzung könnte, wenn der zugehörige private Schlüssel später irgendwie bekannt wird, nachträglich entschlüsselt werden. Aufgrund des Wechsels und "Vergessen" des alten Sitzungsschlüssels ist eine nachträgliche Entschlüsselung der Sitzung nicht oder nur für kleine Abschnitte der Sitzung möglich ("Replay"-Angriff).

## 4) SSH installieren, aktivieren und testen

## 4.1) SSH-Daemon "sshd"

Prüfen, ob SSH installiert ist:

```
rpm -qa | grep "ssh" # --> z.B. "openssh-3.0p1-33 + openssh-askpass-3.0p1-33"
dpkg -l | grep "ssh" # --> z.B. "openssh-server"
```

Auf dem eigenen Rechner muss der SSH-Client "ssh" (oder "Putty") installiert sein, auf dem Remote-Rechner muss ein SSH-Server "sshd" laufen (lauscht auf Port 22). Evtl. Firewalls müssen Verbindungen auf Port 22 erlauben bzw. durchlassen.

SSH-Daemon auf Remote-Rechner (Server) aktivieren:

```
rcsshd start # Temporär oder
/etc/init.d/sshd start # Temporär oder
```

Mar 27, 18 7:53

ssh-HOWTO.txt

Page 4/15

```
initctl ssh start      # Temporär (Upstart) oder
chkconfig -a sshd     # Permanent (-a=add)
```

SSH-Daemon auf Remote-Rechner (Server) deaktivieren:

```
rcsshd stop          # Temporär oder
/etc/init.d/sshd stop # Temporär oder
initctl ssh stop     # Temporär (Upstart) oder
chkconfig -d sshd   # Permanent (-d=del)
```

Überprüfen ob SSH-Daemon "sshd" aktiv ist:

```
/etc/init.d/sshd status # (SysV) oder
initctl ssh status     # (Upstart) oder
service ssh status     # (Systemd) oder
rcsshd status          # (Suse) oder
ps -aux | grep "sshd"  #
```

Bei Änderungen an der SSH-Server-Konfigurations-Datei `"/etc/ssh/sshd_config"` MUSS der SSH-Server neu gestartet und SOLLTEN vorhandene Verbindungen neu aufgebaut werden (bleiben mit alten Einstellungen bestehen):

```
/etc/init.d/sshd restart # (SysV) oder
initctl ssh restart     # (Upstart) oder
service ssh restart     # (Systemd) oder
rcsshd restart          # (Suse)
```

Alternativ kann auch nur die Konfiguration des SSH-Servers neu geladen werden:

```
/etc/init.d/sshd reload # (SysV) oder
initctl ssh reload     # (Upstart) oder
service ssh reload     # (Systemd) oder
rcsshd reload          # (Suse)
```

Zum Testen des SSH-Daemons das Debugging einschalten (verschiedene Stufen), er startet dann nicht als Daemon und nur lx. Seine Fehlermeldungen werden auf dem Fehlerkanal `"stderr"` ausgegeben:

```
sshd -d      # Debug-Meldungen
sshd -dd     # Mehr Debug-Meldungen
sshd -ddd    # Noch mehr Debug-Meldungen (mit sshd_config-Parametern!)
```

Neben BENUTZERSPEZIFISCHEN Schlüsseln gibt es auch RECHNERSPEZIFISCHE Schlüssel, diese werden in der Regel bei der Systeminstallation erzeugt. Sämtliche Schlüssel sollten unbedingt an einer vertraulichen Stelle gesichert werden (z.B. ausgedruckt und in einem verschlossenen Umschlag hinterlegt werden), damit sie bei einem Systemfehler oder einer Neuinstallation wieder hergestellt werden können! Durch sie kann bestimmt werden, auf welche Rechnern überhaupt eine SSH-Verbindung erlaubt ist:

Schlüssel-Datei	Version
/etc/ssh/host_key_rsa	2 (privater RSA2-Schlüssel)
/etc/ssh/host_key_rsa.pub	2 (öffentlicher RSA2-Schlüssel)
/etc/ssh/host_key_dsa	2 (privater DSA-Schlüssel)
/etc/ssh/host_key_dsa.pub	2 (öffentlicher DSA-Schlüssel)
/etc/ssh/host_key_ecdsa	2 (privater ECDSA-Schlüssel)
/etc/ssh/host_key_ecdsa.pub	2 (öffentlicher ECDSA-Schlüssel)
/etc/ssh/host_key_ed25519	2 (privater ED25519-Schlüssel)
/etc/ssh/host_key_ed25519.pub	2 (öffentlicher ED25519-Schlüssel)
/etc/ssh/host_key	1 (privater RSA1-Schlüssel)
/etc/ssh/host_key.pub	1 (öffentlicher RSA1-Schlüssel)

#### 4.2) SSH-Protokoll

Es gibt 2 Protokoll-Versionen von SSH:

Version	Verschlüsselung	Protokoll
SSHv2	RSA, DSA, ECDSA, ED25519	P2
SSHv1	RSA	P1

Sie unterscheiden sich in der Aushandlung des symmetrischen Schlüssels (sicherheitstechnische und patentrechtliche Gründe) und im Protokollablauf (Kommunikation zwischen Client und Server). ACHTUNG: Die Version "SSHv1" sollte aus Sicherheitsgründen nicht mehr verwendet werden:

- 1) Protokoll Version 1 (SSH1)
  - \* Client baut Verbindung zum Server auf Port 22 auf
  - \* Verständigung auf Protokollversion und sonstige Kommunikations-Parameter
  - \* Transfer: Langlebiger Host-Key + kurzleb. Session-Key (1x pro h geändert)
  - \* Bekannt?
  - \* 256-Bit-Session-Key erzeugt, verschlüsselt und an Server übermittelt
  - \* Server entschlüsselt Session-Key
  - \* Session-Key wird für symmetrischen Verschlüsselung benutzt
- 2) Protokoll Version 2 (SSH2)
  - \* Client baut Verbindung zum Server auf Port 22 auf
  - \* Verständigung auf Protokollversion und sonstige Kommunikations-Parameter
  - \* Transfer: Langlebiger Host-Key
  - \* Diffie-Hellman Schlüssel-Vereinbarung --> geheimer Session-Key
  - \* Session-Key wird für symmetrischen Verschlüsselung benutzt

Welche Protokolle ein SSH-Server spricht, ist seiner Antwort auf den Befehl "telnet HOST 22" zu entnehmen (erst Protokoll-Version, dann Implementierung-Version - schlecht, wie vermeiden? z.B. "SSH-2.0-OpenSSH\_6.9"):

```
"SSH-1.5-OpenSSH_..." # --> Nur SSHv1
"SSH-1.99-OpenSSH_..." # --> SSHv1 und SSHv2
"SSH-2.0-OpenSSH_..." # --> Nur SSHv2
```

Die wichtigsten Server-Einstellungen in "sshd\_config":

```
+-----+-----+-----+
| Port          22      | Standard-Port (für interne Anwendung änderbar) |
| Protocol      2       | Probier-Reihenfolge Protokoll-Vers. (nicht 2,1!) |
| ListenAddress 0.0.0.0 | Std: Auf allen NW-Karten lauschen                |
| ForwardX11    yes     | X11-Forwarding aktiv                             |
+-----+-----+-----+
```

#### 4.3) SSH-Client "ssh"

Welche Protokoll-Version für die SSH-Verbindung verwendet wird (ein Server kann beide anbieten), legt der Client beim Verbindungsaufbau fest:

```
ssh -1 ... # Protokoll Version 1
ssh -2 ... # Protokoll Version 2
```

Testen ob ein HOST eine SSH-Verbindung prinzipiell annimmt:

```
telnet HOST 22 # Antwort z.B. "SSH-1.99-OpenSSH_3.0p1"
                # oder          "SSH-2.0-OpenSSH_5.1"
```

Während einer SSH-Anmeldung an einem Remote-Rechner können Kommandos an den SSH-Client abgesetzt werden, die durch das Sonderzeichen "~" eingeleitet werden müssen. Diese "Escape-Sequenzen" dienen zur interaktiven Bedienung der Sitzung und sind nur am Zeilenanfang eingebbar (ansonsten hat "~" normale Bedeutung):

```
+-----+-----+-----+
| ~. | Verbindung abbrechen (alle darüber gemultiplexten Sitzungen) |
| ~B | BREAK an Remote-Rechner senden                                |
| ~C | Kommandozeile öffnen (SSH-Optionen angeben)                  |
| ~R | Neuen Sitzungsschlüssel anfordern (nur SSH2)                  |
| ~^Z | Anhalten (^Z=Strg-Z, mit "fg" weiterlaufen lassen)           |
| ~# | Weitergeleitete Verbindungen auflisten (forwarded)           |
| ~& | In Hintergrund stellen (auf Verb.Ende warten, Tunnel noch genutzt) |
| ~? | Hilfe zu Escape-Sequenzen anzeigen (diese Liste)             |
| ~~ | Tilde-Zeichen selbst eingeben (2x tippen notwendig)          |
+-----+-----+-----+
```

Die wichtigsten Client-Einstellungen in "ssh\_config":

```
+-----+-----+-----+
| Port          22      | Standard-Port (für interne Anwendung änderbar) |
| Protocol      2       | Probier-Reihenfolge Protokoll-Vers. (nicht 2,1!) |
| ForwardX11    yes     | X11-Forwarding aktiv                             |
| TCPKeepAlive  yes     | TCP Keep alive Meldungen senden                 |
| ServerAliveInterval 300 | SSH-Keep alive Meldungen alle 5 Minuten senden |
| EscapeChar    ~       | Escape-Zeichen                                  |
+-----+-----+-----+
```

#### 5) Einsatz von SSH

##### 5.1) Anmelden per SSH-Client

-----  
 Per SSH-Client an Remote-Rechner anmelden (aktueller oder anderer Benutzer):

```
ssh localhost          # Auf eigenem Rechner als aktueller Benutzer
ssh 172.23.19.56       # Als aktueller Benutzer auf Remote-Rechner
ssh -l tom 172.23.10.56 # Als anderer Benutzer (-l=login)
ssh tom@172.23.10.56   # Als anderer Benutzer (@=at)
ssh -p 2222 tom@172.23.10.26 # Auf anderem Port (-p=port)
```

Test des Verbindungs-Aufbaus (-v=verbose, auch mehrfach -vv, -vvv, ...):

```
ssh -v tom@172.23.10.56 # Meldungen zum Verbindungsaufbau
ssh -vv tom@172.23.10.56 # Mehr Meldungen zum Verbindungsaufbau
ssh -vvv tom@172.23.10.56 # Noch mehr Meldungen zum Verbindungsaufbau
```

Beim ERSTEN Verbindungsaufbau zu einem unbekanntem Server, d.h. dessen Host-Schlüssel auf Client-Seite unbekannt ist (nicht in "~/.ssh/known\_hosts" enthalten), folgende Meldung durch Eintippen von "yes" beantworten, um den Host-Schlüssel ohne Verifizierung zu akzeptieren (unsicher!):

```
The authenticity of host '192.168.1.252 (192.168.1.252)' can't be established.
RSA key fingerprint is 8a:58:19:a1:27:95:2b:e4:e2:3e:5f:41:c7:58:2a:d2.
Are you sure you want to continue connecting (yes/no)?
```

ACHTUNG: Text "yes" ist GENAU SO einzutippen und mit <RETURN> abzuschließen, wenn die Verbindung zum unbekanntem Server aufgebaut und sein Host-Schlüssel in "~/.ssh/known\_hosts" eingetragen werden soll.

## 5.2) Server-Authentifizierung

-----  
 Wird die 1. Verbindung zu einem Host mit "yes" (3 Buchstaben!) akzeptiert, dann wird in der lokalen Datei "~/.ssh/known\_hosts" des aktuellen Benutzers der rechner-spezifische ÖFFENTLICHE Schlüssel der Gegenstelle eingetragen (mit Zuordnung zu einer IP):

-----+-----	-----+-----
Schlüsseldatei	Version
-----+-----	-----+-----
/etc/ssh/ssh_host_rsa_key.pub	2 (öffentlicher Schlüssel)
/etc/ssh/ssh_host_dsa_key.pub	2 (öffentlicher Schlüssel)
/etc/ssh/ssh_host_ecdsa_key.pub	2 (öffentlicher Schlüssel)
/etc/ssh/ssh_host_ed25519_key.pub	2 (öffentlicher Schlüssel)
/etc/ssh/ssh_host_key.pub	1 (öffentlicher Schlüssel)
-----+-----	-----+-----

Das ist unsinnig für Rechner, die ständig ihre IP-Adressen ändern. In der Konfigurations-Datei "/etc/ssh/ssh\_config" bzw. "~/.ssh/config" kann daher das Verhalten eingestellt werden:

```
StrictHostKeyChecking Ask          # "Yes" oder "No" möglich (neben "Ask")
```

Die Einstellung "Ask" ist bequem aber auch GEFÄHRLICH, da der Host-Schlüssel kompromittiert sein könnte ("man-in-the-middle"-Angriff). Daher sollte er über einen getrennten Kanal transportiert werden (z.B. USB-Stick, Diskette, CDROM).

Zumindest sollte aus dem übertragenen Host-Schlüssel sein "Fingerprint" gebildet und mit dem des Original-Schlüssels verglichen werden (z.B. am Telefon vorlesen oder von Visitenkarte ablesen).

```
ssh-keygen -l -f KEYFILE # -l=list, -f=file (STD: sha256-Hash)
ssh-keygen -E sha256 -l -f KEYFILE # -l=list, -f=file (sha256-Hash)
ssh-keygen -E md5 -l -f KEYFILE # -l=list, -f=file (md5-Hash)
ssh-keygen -lv -f KEYFILE # -v=verbose --> ASCII-Grafik zusätzlich!
```

Der "Fingerprint" wird von SSH bei der Erstellung eines neuen Schlüssels bzw. bei der ersten Verbindungsaufnahme angezeigt (md5- bzw. sha256-Hash):

```
2048 MD5:da:0a:08:7a:ee:31:40:20:b6:f6:82:df:f3:94:78:84 tsbirn@exeter (RSA)
2048 SHA256:LQu4Q/uhWpU/hrjztIUNdLQu+jnHdqKImQmxsExaP6M tsbirn@exeter (RSA)
```

Die inzwischen ebenfalls verfügbare Darstellung als ASCII-Grafik eignet sich zur groben optischen Kontrolle, ob man mit dem richtigen Rechner verbunden ist:

```
+--[ RSA 2048]-----+
|o.                    |
|+ .                   |
|+                     |
|+ . .                 |
|+. .E . S            |
|.+.o o +              |
|. * = = .            |
```

```
| o o * . |
| .o o |
+-----+
```

### 5.3) Kommando per SSH remote ausführen

Ein Kommando an eine SSH-Verbindung übergeben und ausführen lassen:

```
ssh USER@HOST "CMD"          # Kommando "CMD"
ssh USER@HOST "ls -l"       # Kommando "ls -l"
```

ACHTUNG: Aufpassen bei Metazeichen (resultierenden Befehl ansehen mit "echo"):

```
+-----+
| Befehl          | "*" ausgewertet auf... |
+-----+
| ssh USER@HOST ls -l * | ...lokalem Rechner |
| ssh USER@HOST ls -l \* | ...Remote-Rechner |
| ssh USER@HOST ls -l "*" | ...Remote-Rechner |
| ssh USER@HOST ls -l '*' | ...Remote-Rechner |
| ssh USER@HOST "ls -l *" | ...Remote-Rechner |
| ssh USER@HOST 'ls -l *' | ...Remote-Rechner |
+-----+
```

### 5.4) Dateien per SSH zwischen Rechnern kopieren

- \* ACHTUNG: ":" nicht vergessen, sonst erzeugt man eine LOKALE Kopie!
- \* Standard-Benutzer ist der aktuell angemeldete Benutzer
- \* Standard-Verzeichnisse:
  - + Lokal: Aktuelles Verz. "." des aktuell angemeldeten Benutzers
  - + Remote: Heimat-Verz. des Anmeldebenutzers "~USER"
- \* Passwort wird verlangt, wenn kein Schlüsselaustausch stattgefunden hat
- \* Die Struktur des Filesystems auf dem Remote-Rechner muss bekannt sein, damit man dort mit korrekten Pfaden zugreifen kann
- \* Zugriffsrechte auf Remote-Rechner müssen Zugriff gemäß Anmeldung erlauben

```
scp FILE HOST:          # Remote=Ziel: Heimat-Verz. des aktuellen Users
scp FILE HOST:/tmp     # Remote=Ziel: Verz. "/tmp"
scp FILE USER@HOST:   # Remote=Ziel: Heimat-Verz. von USER ("-l" unmögl!)
scp HOST:FILE .       # Remote=Quelle: "FILE" im Heimat-Verz. (Ziel: akt. Verz.)
```

Man kann sogar zwischen zwei Remote-Rechnern kopieren:

```
scp USER1@HOST1:FILE USER2@HOST2:FILE2 # 2x Schlüsseltausch bzw. anmelden!
```

Einige wichtige Optionen von "scp":

```
+-----+
| Option          | Bedeutung |
+-----+
| -C              | Komprimierung der übertragenen Daten [Compress] |
| -l LIMIT        | Max. Bandbreite in Kbit/s [limit] |
| -o OPTION=VALUE ... | SSH-Option setzen (mehrfach erlaubt) |
| -p             | Zugriffsrechte + -zeiten erhalten [preserve] |
| -P NR          | Port NR statt 22 (GROSS!) [Port] |
| -r             | Rekursiv (alle Verz. + ihre Inhalte) |
+-----+
| -q             | Weniger Meldungen (Fortschrittanzeige) [quiet] |
| -v             | Ablaufmeldungen [verbose] |
| -vv           | Mehr Ablaufmeldungen |
| -vvv          | Noch mehr Ablaufmeldungen |
+-----+
```

### 5.5) X Window-Protokoll tunneln

X Window-Protokoll vom Server zum Client durchtunneln  
(Variable "DISPLAY" anschließend gesetzt):

```
ssh -X 172.23.10.56      # Erlauben (-X=enable)
ssh -x 172.23.10.56     # Verboten (-x=disable)
ssh -Y 172.23.10.56     # Erlauben (-Y=enable)
```

Um X Window-Anwendungen remote auf dem Remote-Server zu starten und dann lokal auf dem Client anzuzeigen, müssen verschiedene Bedingungen erfüllt sein:

- \* Remote-Server muss das unterstützen  
(Eintrag "X11Forwarding" in "/etc/ssh/sshd\_config", Std. meist "no").
- \* Lokaler Client muss das unterstützen (Var. DISPLAY muss gesetzt sein,

Eintrag "ForwardX11" bzw. "ForwardX11Trusted" in "/etc/ssh/ssh\_config" bzw. "~/.ssh/config", Std. meist "no"; oder Schalter "-X" (enable) bzw. "-Y" (enable trusted) beim ssh-Aufruf mitgeben).

\* Evtl. "xhost +HOST" auf lokalem Rechner (macht eigentlich "xauth")

### 5.6) Schlüsselbasierte Client-Authentifizierung einrichten (kein Passwort)

Bisher: Bei der Verbindungsaufnahme authentifiziert sich der Server beim Client und der Client muss zur Authentifizierung sein Passwort eingeben (diese Kommunikation wird bereits verschlüsselt).

Neu: Die Authentifizierung des Clients soll OHNE Passworteingabe erfolgen. Dazu muss für den anzumeldenden Benutzer ein Schlüssel generiert werden, dessen öffentlicher Teil auf dem Remote-Rechner (Server) in die Datei "~/.ssh/authorized\_keys" einzutragen ist (in seinem Heimatverz.).

Ein Benutzer hat zunächst kein Verz. "~/.ssh/" in seinem Heimat-Verz. Dieses wird erzeugt, wenn für ihn ein Schlüsselpaar generiert wird (ohne "Passphrase" = leere Passphrase, -t=type, "\*.pub" = öffentlicher Schlüssel):

Aufruf	Erzeugte Schlüsseldateien
ssh-keygen -t rsa	~/.ssh/id_rsa + ~/.ssh/id_rsa.pub
ssh-keygen -t dsa	~/.ssh/id_dsa + ~/.ssh/id_dsa.pub
ssh-keygen -t ecdsa	~/.ssh/id_ecdsa + ~/.ssh/id_ecdsa.pub
ssh-keygen -t ed25519	~/.ssh/id_ed25519 + ~/.ssh/id_ed25519.pub
ssh-keygen -t rsa1	~/.ssh/identity + ~/.ssh/identity.pub
ssh-keygen	(analog)

Man erhält einen "Fingerprint" des generierten Schlüssels ausgegeben, den man z.B. auf seiner Web-Seite veröffentlicht oder in seine Visitenkarte einträgt. Andere können damit verifizierten, ob der Verbindungspartner wirklich der richtige ist.

```
2048 MD5:da:0a:08:7a:ee:31:40:20:b6:f6:82:df:f3:94:78:84 tsbirn@exeter (RSA)
2048 SHA256:LQu4Q/uhWpU/hrjztIJNdLQu+jnHdqKTmQmxsExaP6M tsbirn@exeter (RSA)
```

Der Fingerprint kann auch optisch durch eine ASCII-Grafik dargestellt werden:

```
+--[ RSA 2048]-----+
|o.                    |
|+ .                   |
|+                     |
|+ . .                 |
|+. .E . S             |
|.+.o o +              |
|. * = = .             |
| o o * .              |
|.o o                  |
+-----+

```

Nachträglich Fingerprint bzw. ASCII-Grafik zu einem Schlüssel anzeigen:

```
ssh-keygen -l           -f KEYFILE      # -l=list, -f=file (STD: sha256-Hash)
ssh-keygen -l -E md5   -f KEYFILE      # md5-Hash
ssh-keygen -l -E sha256 -f KEYFILE      # sha256-Hash
ssh-keygen -lv         -f KEYFILE      # -v=verbose --> ASCII-Grafik zusätzlich!
```

Der Weg, auf dem der öffentliche Schlüsselteil anschließend zum jeweiligen Remote-Rechner transportiert wird, darf nicht manipulierbar sein (z.B. per Hand, Telefon, Diskette, USB-Stick, NICHT per eMail!). Eine weitere (unsichere) Möglichkeiten wäre der Transport per Maus in einer SSH-Anmeldung.

```
cat ~/.ssh/id_rsa.pub          # Öffentlichen Client Schlüssel anzeig.
ssh USER@HOST                 # Remote-Anmeldung per Passwort
mkdir ~/.ssh                   # Unterverz. für SSH-Keys anlegen
vi ~/.ssh/authorized_keys      # Datei mit Client-Schlüsseln anzeigen
... angezeigten Schlüssel per Maus als oberste Zeile einfügen ...
:wq                             # Datei speichern (w=write, q=quit)
exit                           # Remote-Abmeldung
ssh USER@HOST                 # Anmeldung per Schlüsselaustausch
```

Oder mit "scp":

```
scp ~/.ssh/id_rsa.pub USER@HOST:/tmp # Anmeldung per Passwort
ssh USER@HOST                       # Anmeldung per Passwort
mkdir ~/.ssh                          # Unterverz. für SSH-Keys anlegen
```



```
cat /tmp/id_rsa.pub >> ~/.ssh/authorized_keys # Öffentl. Client-Key anhängen
exit # Remote-Abmeldung
ssh USER@HOST # Anmeldung per Schlüsselaustausch
```

Oder alles auf einen Schlag:

```
ssh USER@HOST "mkdir ~/.ssh; # Anmeldung per Passwort
cat >> ~/.ssh/authorized_keys" < ~/.ssh/id_rsa.pub
```

Oder per Skript:

```
ssh-copy-id USER@HOST # ~/.ssh/id_rsa.pub (Std)
ssh-copy-id -i USER@HOST # ~/.ssh/identity.pub
ssh-copy-id -i KEYFILE USER@HOST # FILE
```

ACHTUNG: Kein Anfangs/Endstück des öffentlichen Schlüssels weglassen, muss aus EINER Zeile bestehen (keine Zeilenumbrüche).

Nun sollte ein Anmeldung per "ssh" OHNE Passwordeingabe möglich sein!

### 5.6.1) Grafik zum Ablauf einer Client-Authentifizierung

Letzten Endes hat folgender Austausch des öffentlichen Schlüsselteils der jeweiligen Gegenstelle "über Kreuz" stattgefunden:

USER1	ANMELDUNG	USER2
ssh USER2@HOST	=====>	sshd
+-----+   Client     (local)   +-----+	SSH-Protokoll	+-----+   Server (HOST)     (remote)   +-----+
(MANUELL erzeugt: ssh-keygen -t rsa) /home/USER1/.ssh/id_rsa.pub		(bei Installation AUTOMAT. erzeugt) /etc/ssh/ssh_host_rsa_key.pub
+-----+   (AUTOMATISCH bei 1. SSH-Verb. + "yes")   +-----+		+-----+
+-----+   (MANUELL pro Benutzer)   +-----+		+-----+
v		v
/home/USER1/.ssh/known_hosts # oder /etc/ssh/ssh_known_hosts		/home/USER2/.ssh/authorized_keys (/etc/ssh/authorized_keys NICHT!)

TIPP: Sammeln der bei allen Benutzern (inkl. "root") vorh. "~/.ssh/known\_hosts" in zentraler Datei "ssh\_known\_hosts" (doppelte Einträge nur 1x):

```
cat /home/*/.ssh/known_hosts > /tmp/uuu # Umlenken + sudo nicht glz.
sudo cp /root/known_hosts /tmp/rrr #
cat /tmp/uuu /tmp/rrr | sort | uniq > /tmp/hhh # Umlenken + sudo nicht glz.
sudo cp /tmp/hhh /etc/ssh/ssh_known_hosts #
```

HINWEIS: Ob das eine gute Idee ist, sei dahingestellt! Die zentralen Schlüssel in "/etc/ssh/ssh\_known\_hosts" sollten VERIFIZIERT sein. Benutzerspezifische Schlüssel in "~/.ssh/known\_hosts" könnten durch Eingabe von "yes" bei der ERSTEN Verbindungsaufnahme akzeptiert sein und sind dann NICHT verifiziert.

NACH dem Austausch der öffentlichen Schlüssel wird häufig in "/etc/ssh/sshd\_config" die Authentifizierung via Passwort deaktiviert (Std = vordefinierte Defaultwerte, Sec = sichere Werte):

Konfigurationsvariable	Std	Sec	Bedeutung
RSAAuthentication	yes	yes	Reine RSA-Authentifiz. erlaubt (nur P1)
PubkeyAuthentication	yes	yes	Public-Key Authentifiz. erlaubt (nur P2)
PasswordAuthentication	yes	no	SSH fragt selber nach dem Passwort
UsePAM	no	no	Sonst wird Passwort von PAM verlangt

Sicherheitstechnisch wichtige Einstellungen in "/etc/ssh/sshd\_config":

Konfigurationsvariable	Std	Sec	Bedeutung
PermitRootLogin	yes	no	Direkte root-SSH-Anmeldung nicht möglich
PermitEmptyPasswords	no	no	Leere Passworte verboten
IgnoreRhosts	yes	yes	"~/.rhosts" "~/.shosts" nicht benutzen
IgnoreUserKnownHosts	no	yes	Userspez. "~/.ssh/known_hosts" verboten, Systemspez. "/etc/ssh/known_hosts" nötig

```
+-----+-----+-----+-----+
Weitere sinnvolle Einstellungen in "/etc/ssh/sshd_config":
```

Konfigurationsvariable	Std	Sec	Bedeutung
StrictModes	yes	yes	Zugriffsrechte auf Dateien prüfen
X11Forwarding	no	no	X11-Protokoll tunneln
X11DisplayOffset	10	10	X11-Display Startnummer
TCPKeepAlive	yes	yes	TCP-Keepalive Meldungen schicken
Port	122	22	Port 22 --> 122 (gegen Bot-Attacken)
Protocol	2,1	2	Protokoll 1 ist unsicher
ChallengeResponseAuthentication	yes	no	CR-Authentifizierung erlaubt (PAM)

#### 5.7) Privaten Schlüssel mit Passphrase sichern

Der private Schlüssel sollte durch eine "Passphrase" (das ist ein längerer Text ähnlich einem Passwort) gesichert werden, mit der er "überschlüsselt" wird, damit er nicht im Klartext gespeichert ist. Diebstahl oder Kompromittierung des Rechners führt dann nicht zur Kompromittierung des privaten Schlüssels. Beim Remote-Login per SSH wird dann nach der Passphrase des Schlüssels gefragt (damit dieser entschlüsselt werden kann) statt nach dem Passwort.

```
$ ssh-keygen -p -f ~/.ssh/id_dsa      # -p=passphrase, -f=file
$ ssh-keygen -p                      # Automatisch Datei ~/.ssh/id_rsa
Enter file in which the key is (/home/tsbirn/.ssh/id_rsa):
Enter old passphrase: XXXXXXXXXX
Key has comment '/home/tsbirn/.ssh/id_rsa'
Enter new passphrase (empty for no passphrase): YYYYYYYYYY
Enter same passphrase again: YYYYYYYYYY
Your identification has been saved with the new passphrase.
```

Die "Passphrase" ist jedesmal einzugeben, wenn ein Zugriff auf den privaten Schlüssel notwendig ist (für automatischen Verbindungs-Aufbau ist das unsinnig, man kann aber auch mehr als ein Schlüsselpaar erzeugen, um dies zu umgehen).

Soll die "Passphrase" wieder entfernt werden, den privaten Schlüssel analog mit einer LEEREN Passphrase überschlüsseln.

#### 5.8) SSH-Agent zum Schlüssel verwalten

Alternativ kann die Passphrase über einen SSH-Agenten zur Verfügung gestellt werden, so dass sie nur 1x eingegeben werden muss, anschließend im Speicher steht und bei jedem SSH-Kommando automatisch zur Verfügung gestellt wird:

```
ssh-agent # Linux
Pageant  # Windows (Putty Agent)
```

Er dient als "Wrapper" für beliebige andere Prozesse (z.B. die Shell oder den X-Server) und hält für sie entschlüsselte private Schlüssel im Speicher vor.

```
ssh-agent # SSH-Agenten starten --> ergibt Shell-Befehle
eval `ssh-agent -c` # -c=C-Shell Befehle erzeugen (C-Shell Syntax)
eval `ssh-agent -s` # -s=Bourne-Shell Befehle erzeugen (Bourne-Syntax)
eval $(ssh-agent -s) # -s=Bourne-Shell Befehle erzeugen (Bash-Syntax)
```

Der Agent produziert als Ausgabe Shell-Befehle, die zwei Umgebungs-Variablen SSH\_AUTH\_SOCKET und SSH\_AGENT\_PID setzen, mit deren Hilfe andere Prozesse den Agenten finden und mit ihm kommunizieren können:

```
SSH_AUTH_SOCKET=/tmp/ssh-qsii28019/agent.28019; export SSH_AUTH_SOCKET;
SSH_AGENT_PID=28020; export SSH_AGENT_PID;
echo Agent pid 28020;
```

Diese Ausgabe auf Stdout muss abgefangen und als Befehl ausgeführt werden, daher die Kommando-Substitution \$(...) bzw. `...` und Auswertung per "eval".

Nach dem Start des SSH-Agenten werden SSH-Schlüssel an ihn übergeben durch:

```
Pageant-Icon # Windows, Passphrase 1x eingeben (in graf. Dialog)
ssh-add      # Linux, Passphrase 1x eingeben (auf Kommandozeile)
ssh-add < /dev/null # Linux, Passphrase 1x eingeben (in graf. Dialog)
```

Dabei wird wenn notwendig 1x die Passphrase zum Entschlüsseln abgerufen.

```
ssh-add # Alle verfügbaren Schlüssel merken (max. 3)
```

Mar 27, 18 7:53

## ssh-HOWTO.txt

Page 11/15

```
ssh-add ~/.ssh/id_rsa      # Einen RSA-Schlüssel merken
ssh-add ~/.ssh/id_dsa      # Einen DSA-Schlüssel merken
ssh-add ~/.ssh/id_ecdsa    # Einen ECDSA-Schlüssel merken
ssh-add ~/.ssh/id_ed25519  # Einen ED25519-Schlüssel merken
ssh-add ~/.ssh/identity    # Einen RSA-Schlüssel merken
```

TIPP: Folgenden Alias "sshadd" definieren, der die beiden notwendigen Befehle kombiniert und ihre etwas komplexe Syntax verpackt:

```
alias sshadd="eval $(ssh-agent -s); ssh-add"      # oder
alias sshadd="eval $(ssh-agent -s); ssh-add < /dev/null" #
```

### 5.9) Bei SSH-Anmeldung Kommando ausführen/Optionen setzen

In Datei "~/.ssh/authorized\_keys" können am Zeilenanfang jedes Schlüssels VOR dem Schlüssel-Typ "ssh-rsa/ssh-dsa/ssh-ecdsa/ssh-ed25519/ssh" folgende Optionen angegeben werden, die dann bei einer Anmeldung über diesen Schlüssel gelten:

Option	Bedeutung
command="CMD...;..."	Kommando ausführen (mehrere durch ";" trennen)
environment="VAR=VALUE"	Umgebungsvariable setzen (durch "," trennen)
from="HOSTLIST"	Auf bestimmte Remote-Rechner beschränken (" , ")
no-agent-forwarding	SSH-Agenten NICHT weitergeben
no-port-forwarding	Ports NICHT weiterleiten (-L/-R)
no-pty	Kein Terminal zuweisen
no-user-rc	"~/.ssh/rc" NICHT ausführen
no-X11-forwarding	X11 NICHT weiterleiten
permitopen="HOST:PORT"	Lokale Weiterleitung beschränken
tunnel="N"	Tunnel N erzwingen (sonst nächstes freies)

ACHTUNG: Alle Optionen in eine Zeile schreiben, Werte mit Leerzeichen in "... " oder '...' einschließen, mehrere Angaben durch "," trennen (keine Leerzeichen zur Trennung verwenden), mehrere Kommandos durch ";" trennen".

Beispiele:

```
command="/bin/ls > /tmp/ls.txt"      ssh-rsa  KEY...  # Nur Kmdo ausführen + sofort wieder abm
elden
command="/usr/bin/ssh tom@192.168.1.1" ssh-dsa  KEY...  # Auf weiteren Rechner springen
from="192.168.1.1"                  ssh-ecdsa KEY...  # Nur von diesem Rechner Anmeldung erlau
bt
from="192.168.1.1",command="/usr/bin/ssh tom@192.168.1.1" ssh KEY...  # Auf weiteren Rechner spri
ngen
```

Der Eintrag "command=CMD..." sorgt dafür, dass kein echter Login mehr erfolgt, sondern das Kommando "CMD..." ausgeführt und danach die SSH-Verbindung wieder beendet wird (sofern eine schlüsselbasierte Anmeldung erfolgreich stattfindet). Bei einer passwortbasierten Anmeldung wird kein Kommando ausgeführt. Sinn:

- \* Kommando/Skript per reiner Anmeldung ausführen (freischalten)
- \* SSH-Verbindung nach außen über Proxy-Server (keine End-zu-End-Verbindung)
- \* SSH-Datenfluss überwachen

HINWEIS: Die Dokumentation dazu findet man in "man sshd" (Daemon!)

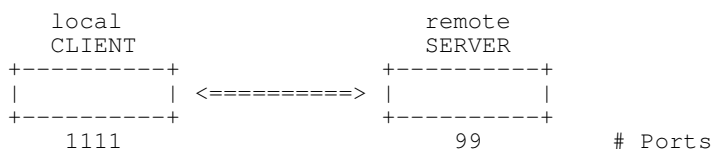
### 6) Port-Forwarding (Tunneling von Protokollen)

Durchschleusen von anderen Protokollen durch einen SSH-Tunnel. Insbesondere KLARTEXT-Protokolle (z.B. TELNET, FTP, POP3, IMAP, SMTP, SNMP, ...) können auf diese Weise nachträglich per verschlüsselten "Wrapper" abgesichert werden.

Dazu muss folgende "sshd"-Einstellung in "/etc/ssh/sshd\_config" gelten:

```
AllowTcpForwarding yes
```

Es gibt zwei "Richtungen" des "Tunneling" gesteuert durch das SSH-Kommando auf dem CLIENT, je nachdem ob anschließend eine Verbindung durch den Tunnel ausgehend vom lokalen Rechner CLIENT (-L/-D) oder ausgehend vom remote Rechner SERVER (-R) aufgebaut werden soll:



1. Verbindung CLIENT --> SERVER öffnen,  
 lokal am CLIENT auf Port 1111 lauschen,  
 Daten weitergeben an SERVER auf Port 99  
 (d.h. CLIENT ist Client, SERVER ist Server).  
 ERGEBNIS: Tunnel von CLIENT:1111 --> SERVER:99  
 ACHTUNG: "localhost" oder "127.0.0.1" statt CLIENT-IP nach -L!

```
CLIENT: ssh -L 1111:localhost:99 SERVER # Tunnel aufbauen (-L=local)
CLIENT: telnet localhost 1111 # Tunnel CLIENT --> SERVER nutzen
```

2. Verbindung SERVER --> CLIENT öffnen,  
 remote am SERVER auf Port 99 lauschen,  
 Daten weitergeben an CLIENT auf Port 1111  
 (d.h. SERVER ist Client, CLIENT ist Server!).  
 ERGEBNIS: Tunnel von SERVER:99 --> CLIENT:1111  
 ACHTUNG: CLIENT-IP statt "localhost" nach -R!

```
CLIENT: ssh -R 99:CLIENT:1111 SERVER # Tunnel aufbauen (-R=remote)
SERVER: telnet localhost 99 # Tunnel SERVER --> CLIENT nutzen
```

## Beispiele:

A) Lokal von Port 8080 auf Port 80 eines remote SERVER tunneln  
 (keiner kann Kommunikation mitlesen, -L=lokal, lange laufendes Kommando  
 starten, das keine Rechenzeit verbraucht und es in den Hintergrund schieben):

```
ssh -L 8080:localhost:80 USER@SERVER "sleep 100000" & # Nur wenn ohne Pw. mögl.
ssh -L 8080:localhost:80 USER@SERVER "sleep 100000" # Auch mit Passwortabfrage
Strg-Z
bg
```

Im Browser auf CLIENT den lokalen Rechner auf Port 8080 ansprechen,  
 die HTTP-Daten werden verschlüsselt an/vom SERVER Port 80 übertragen:

```
http://localhost:8080 # <-> SERVER:80
```

B) POP3-Protokoll lokal von Port 5000 auf remote SERVER Port 110 tunneln  
 (keiner kann Kommunikation mitlesen):

```
ssh -L 5000:localhost:110 SERVER # Tunnel aufbauen
fetchmail # Gültige ".fetchmailrc" muss existieren
# und auf localhost:5000 zugreifen
```

C) Verschlüsselte Telnet-Verbindung lokal von Port 5000 auf remote SERVER  
 Port 23 (telnet) aufbauen (vorher telnet-Daemon/Server auf SERVER installieren  
 und aktivieren sowie "xinetd" rekonfigurieren und neu starten):

```
ssh -L 5000:localhost:23 SERVER "sleep 100000" # Tunnel aufbauen
telnet localhost 5000 # Telnet-Verb. aufbauen
```

D) Von Port 3306 lokal auf Port 3306 remote tunneln (MySQL-Server):

```
ssh -L 3306:localhost:3306 SERVER # Tunnel aufbauen
mysql -uUSER -p --protocol tcp localhost # TCP-Verbindung erzwingen!
```

E) Port 80 auf Zielrechner SERVER per Zwischenrechner ZWHOST vom CLIENT aus  
 auf Port 10080 verbinden (2 Tunnels!) und somit das HTTP-Protokoll (Port 80)  
 vom SERVER zum Port 10080 auf dem CLIENT über eine Zwischenstation ZWHOST  
 (z.B. Firewall) tunneln:

```

CLIENT                                ZWHOST                                SERVER
+-----+                             +-----+                             +-----+
|         | <=====> |         |         | <=====> |         |
+-----+                             +-----+                             +-----+
10080                                     (3080)                                     80
```

Variante 1):  
 ssh -l root -L 10080:SERVER:80 ZWHOST # Auf ZWHOST ausgeführt

Variante 2):  
 ssh -l root -L 10080:localhost:3080 ZWHOST \ # Auf CLIENT ausgeführt  
 "ssh -L 3080:localhost:80 SERVER" # Auf ZWHOST ausgeführt

Ablauf: Anmelden am SSH-Port von Rechner ZWHOST  
 Tunneln von Port 10080 am eigene Rechner LOCALHOST  
 zu Port 3080 auf mittlerem Rechner ZWHOST (-L = "localhost" = dort)  
 Anmelden am SSH-Port von Rechner SERVER  
 Tunneln von Port 3080 am Rechner ZWHOST  
 zu Port 80 auf Endrechner SERVER (-L = "localhost" = dort)

HINWEIS: Damit obiger Befehl nicht ständig einzutippen ist, sollte er per

Alias oder Funktion in einer der persönlichen Shell-Konfigurations-Dateien oder per Shell-Skript in persönlichen Binär-Verz. realisiert werden:

```
# Alias
alias srvhttp='ssh -l root -L 10080:localhost:3080 ZWHOST \
                "ssh -L 3080:localhost:80  SERVER"'

# Funktion
srvhttp()
{
    ssh -l root -L 10080:localhost:3080 ZWHOST \
        "ssh -L 3080:localhost:80  SERVER"
}

# Shell-Skript "srvhttp.sh" (ausführbar machen: chmod a+x srvhttp.sh):
+-----+
|#!/bin/sh|
|ssh -l root -L 10080:localhost:3080 ZWHOST \|
|      "ssh -L 3080:localhost:80  SERVER" |
+-----+
```

F) Konfiguration einer "Putty-Session" am Windows-Client 192.168.0.200, die über den Zwischenrechner 192.168.0.201 auf den Web-Server 192.168.0.250 Web-Zugriff durchführen soll:

```
192.168.0.200      192.168.0.201      192.168.0.250
User: Admin        User: schulung      User: kurs1
+-----+          +-----+          +-----+
|  WIN  | <===== > | ZWHOST | <===== > | SERVER |
+-----+          +-----+          +-----+
      80              3080              80
```

In "Putty-Session" vorzunehmende Einstellungen:

```
Session --> Host Name (or IP address):          # Passwort/Passphrase
schulung@192.168.0.201
Connection --> SSH --> Tunnels --> Forwarded Ports: # Source port: 80
L80 localhost:3080                               # Destination: localhost:3080
Connection --> SSH --> Remote Command:
ssh -l kurs1 -L 3080:localhost:80 192.168.0.250 # Passwort für "kurs1"!
ssh -L 3080:localhost:80 kurs1@192.168.0.250   # (analog)
```

Achtung: Wahrscheinlich ist 2x Passwort/Passphrase einzugeben:  
 1) Erst bei Verbindung Client --> Zwischenrechner für User "schulung".  
 2) Dann bei Verbindung Zwischenrechner --> Server für User "kurs1".

## 7) Übersicht

### 7.1) Kommandos

```
ssh, slogin      Verbindung zu fremdem Rechner aufnehmen
                  (sicherer Ersatz von "telnet", "rsh", "rlogin")
scp              Dateien von/zu fremden Rechner transferieren
                  (sicherer Ersatz von "rcp")
sftp             Dateien von/zu fremden Rechner transferieren
                  (sicherer Ersatz von "ftp")
ssh-keygen       Schlüsselpaar erzeugen
ssh-keyscan      Öffentlichen Schlüssel von Rechnern holen
ssh-add          Privaten Schlüssel beim "ssh-agent" registrieren
ssh-agent        Privaten Schlüssel verwalten
                  (automatische Beantwortung von "challenges")
```

### 7.2) Server-Konfigurations-Dateien

```
/etc/ssh/sshd_config      Zentrale Server-Konfiguration
                           (es gibt keine benutzerspezifische!)

/etc/ssh/ssh_host_rsa_key  Server Identity (Private Key, Protocol V2 = RSA)
/etc/ssh/ssh_host_rsa_key.pub  Server Identity (Public Key, Protocol V2 = RSA)
/etc/ssh/ssh_host_dsa_key    Server Identity (Private Key, Protocol V2 = DSA)
/etc/ssh/ssh_host_dsa_key.pub  Server Identity (Public Key, Protocol V2 = DSA)
/etc/ssh/ssh_host_ecdsa_key  Server Identity (Private Key, Protocol V2 = ECDSA)
/etc/ssh/ssh_host_ecdsa_key.pub  Server Identity (Public Key, Protocol V2 = ECDSA)
/etc/ssh/ssh_host_ed25519    Server Identity (Private Key, Protocol V2 = ED25519)
/etc/ssh/ssh_host_ed25519_key.pub  Server Identity (Public Key, Protocol V2 = ED25519)
/etc/ssh/ssh_host_key        Server Identity (Private Key, Protocol V1 = RSA)
/etc/ssh/ssh_host_key.pub    Server Identity (Public Key, Protocol V1 = RSA)

/etc/ssh/ssh_known_hosts    Public Keys der bekannten Remote-Server (fest)
```

## 7.3) Client-Konfigurations-Dateien

```
-----
/etc/ssh/ssh_config      Zentrale Client-Konfiguration
~/.ssh/config           Benutzerspezifische Client-Konfiguration

~/.ssh/id_rsa           User Identity (Private Key, Protocol 2 = RSA)
~/.ssh/id_rsa.pub       User Identity (Public Key, Protocol 2 = RSA)
~/.ssh/id_dsa           User Identity (Private Key, Protocol 2 = DSA)
~/.ssh/id_dsa.pub       User Identity (Public Key, Protocol 2 = DSA)
~/.ssh/id_ecdsa         User Identity (Private Key, Protocol 2 = ECDSA)
~/.ssh/id_ecdsa.pub     User Identity (Public Key, Protocol 2 = ECDSA)
~/.ssh/id_ed25519       User Identity (Private Key, Protocol 2 = EC25519)
~/.ssh/id_ed25519.pub   User Identity (Public Key, Protocol 2 = EC25519)
~/.ssh/identity         User Identity (Private Key, Protocol 1 = RSA)
~/.ssh/identity.pub     User Identity (Public Key, Protocol 1 = RSA)

~/.ssh/known_hosts      Public Keys der bekannten Remote-Server (updatebar)
~/.ssh/authorized_keys  Public Keys der bekannten Remote-Benutzer
```

## 7.4) Login-Shell Konfigurations-Dateien

```
-----
/etc/ssh/sshr           Zentrale Kommandos für Login-Shells
~/.ssh/rc               Benutzerabh. Kommandos für Login-Shell
~/.ssh/environment      Environment-Variablen für Login-Shell
```

## 8) Tipps

\* Öffentlichen Schlüssel per Kommandoaufruf vom Remote-Rechner holen:

```
ssh-keyscan -t rsa HOST      # V2: RSA-Schlüssel von HOST holen
ssh-keyscan -t dsa HOST     # V2: DSA-Schlüssel von HOST holen
ssh-keyscan -t ecdsa HOST   # V2: ECDSA-Schlüssel von HOST holen
ssh-keyscan -t ed25519 HOST # V2: ED25519-Schlüssel von HOST holen
ssh-keyscan HOST           # V1: RSA-Schlüssel von HOST holen
```

\* Alle Rechner in Datei "ssh\_hosts" finden, die einen neuen oder anderen öffentlichen RSA/DSA/ECDSA/ED25519-Schlüssel haben als in "ssh\_known\_hosts" hinterlegt (beide Dateien müssen nicht sortiert sein):

```
ssh-keyscan -t rsal,rsa,dsa,ecdsa,ed25519 -f ssh_hosts |
sort -u - ssh_known_hosts |
diff - <(sort ssh_known_hosts)
```

\* Änderungen an der SSH-Server-Konfigurations-Datei "/etc/ssh/sshd\_config" wirken sich erst dann aus, wenn der SSH-Daemon neu gestartet wird.

\* Änderungen an der SSH-Client-Konfigurations-Datei "/etc/ssh/ssh\_config" bzw. "~/.ssh/config" wirken sich automatisch auf DANACH neu hergestellte Verbindungen aus, NICHT aber auf bereits aktive Verbindungen.

\* Der eigene Benutzername ist Standard bei der SSH-Anmeldung, sofern man keinen anderen angibt (per -l USER oder USER@HOST). D.h. man kann sich mit einem anderen Benutzernamen auf dem Remote-Rechner anmelden, sofern man dessen Passwort weiß oder dessen privaten Key besitzt.

\* Besitzverhältnisse + Zugriffsrechte der SSH-Konfigurations-Dateien und ihrer Verz.pfade müssen einige Bedingungen erfüllen, damit SSH eine schlüsselbasierte Anmeldung erlaubt (falls "StrictModes yes" in "/etc/ssh/sshd\_config" gesetzt ist):

- + Besitzer von "/home/USER" + "/home/USER/.ssh" + "~/.ssh/authorized\_keys" muss identisch sein zum sich per SSH anmeldenden Benutzer USER.
- + "w" "Besitzer-Gruppe" + "Andere": darf nicht gesetzt sein
- + "r" "Besitzer-Gruppe" + "Andere": darf bei priv. Schlüssel nicht ges. sein

Typische Fehlermeldungen in "/var/log/auth.log" (bzw. "/var/log/messages") auf der Server-Seite bei falschen Zugriffsrechten auf Konfigurations-Dateien:

```
Authentication refused: bad ownership or modes for directory /home/USER1
Authentication refused: bad ownership or modes for directory /home/USER1/.ssh
```

\* Der "ssh-agent" legt die Passphrase bzw. den privaten Schlüssel NIE auf Datei ab, sondern behält sie im Speicher, bis der Benutzer sich wieder abmeldet.

\* Der "ssh-agent" ist nur für Programme aktiv, die im gleichen Terminal wie er gestartet werden (die gleiche Shell als Elternprozess haben).

\* Standardmäßig erfolgt keine X11-Umlenkung mehr. Dazu ist die Option "-X/-Y" beim Aufruf von "ssh" anzugeben oder "ForwardX11" und "TrustedForwardX11" in "/etc/ssh/ssh\_config" bzw. "~/.ssh/config" auf "yes" zu setzen.

- \* Zum testen von Verbindungsproblemen:
  - ssh -v # Meldungen zum Verbindungsaufbau
  - ssh -vv # Mehr Meldungen zum Verbindungsaufbau
  - ssh -vvv # Noch mehr Meldungen zum Verbindungsaufbau
- \* Woher stammt der HOST-Key unter "/etc/ssh"?
  - Wird beim Installieren eines Rechners automatisch eingerichtet.
  - > Kopien eines Rechners als virtuelle Maschine haben gleichen HOST-Key!
- \* Unbedingt Passphrase != Passwort(e) wählen
  - (höhere Sicherheit; weniger Verwirrung: was mache ich eigentlich?).
- \* Privater Schlüssel:
  - + DIREKT auf dem Rechner erzeugen, wo er eingesetzt werden soll
  - + NIE weitergeben/kopieren/...
  - + ÜBERSCHLÜSSELN mit Passphrase falls Rechner allgemein zugänglich ist
  - + SICHER verwahren: Bei Verlust sind alle mit dem öffentlichen Schlüssel verschlüsselten Daten verloren (z.B. ausgedruckt in Safe legen)
- \* Typischer FEHLER: Neues Schlüsselpaar erzeugen, dabei wird altes überschrieben
  - > Schlüsselpaar ausdrucken und sicher verwahren
- \* In Datei "authorized\_keys" gilt:
  - + Jedes einzelne Zeichen wichtig
  - + Anzahl der Leerzeichen wichtig
  - + Vorne Einschränkungen für die Verbindung eintragbar (bis 1. Leerzeichen)
  - + Text hinten (USER@HOST) irrelevant (bel. Kommentar nach letztem Leerz.)
  - + Pro Zeile ein vollständiger (langer) Eintrag
  - + Leerzeilen werden ignoriert

#### 9) Links

- |   |                                    |
|---|------------------------------------|
| -----   |                                    |
| * <a href="http://www.ssh.fi">http://www.ssh.fi</a>   | SSH (original)                     |
| * <a href="http://www.openssh.com">http://www.openssh.com</a>   | OpenSSH                            |
| * <a href="http://www.openssh.org">http://www.openssh.org</a>   | OpenSSH                            |
| * <a href="ftp://ftp.de.openbsd.org/pub/unix/OpenBSD/OpenSSH/portable">ftp://ftp.de.openbsd.org/pub/unix/OpenBSD/OpenSSH/portable</a> | OpenSSH                            |
| * <a href="http://www.chiark.greenend.org.uk/~sgtatham/putty">http://www.chiark.greenend.org.uk/~sgtatham/putty</a>                   | Putty (Windows SSH)                |
| * <a href="ftp://ftp.freeware.com/pub/infozip/zlib">ftp://ftp.freeware.com/pub/infozip/zlib</a>                                       | Zlib                               |
| * <a href="http://www.openbsd.org">http://www.openbsd.org</a>   | OpenBSD                            |
| * <a href="http://www.openssl.org">http://www.openssl.org</a>   | OpenSSL                            |
| * <a href="http://www.winscp.net">http://www.winscp.net</a>   | WinSCP                             |
| * <a href="http://www.mobatek.net">http://www.mobatek.net</a>   | MobaXterm (X-Terminal für Windows) |
| * <a href="http://www.bananahut.net">http://www.bananahut.net</a>   | MobaXterm Plugins                  |
| * <a href="http://www.filezilla.de">http://www.filezilla.de</a>   | FileZilla                          |
| * <a href="http://kitty.9bis.com">http://kitty.9bis.com</a>   | KiTTY (Putty-Ersatz)               |