

HOWTO zu SSH - Secure Shell
 (C) 2007-2011 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>
 OSTC GmbH, <http://www.ostc.de>
 \$Id: ssh-HOWTO.txt,v 1.22 2011-03-11 08:48:47 tsbirn Exp \$

Dieses Dokument beschreibt Grundlagen und Einsatz von SSH unter UNIX/Linux.

Inhaltsverzeichnis

- 1) Einführung
- 2) Übersicht Konfigurationsdateien
- 3) Prinzipielle Arten von Verschlüsselung
- 4) SSH installieren, aktivieren und testen
- 5) Anwendungen von SSH
 - 5.1) Anmelden per SSH
 - 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) Client-Authentifizierung (ohne Passwort)
 - 5.7) Privaten Schlüssel mit Passphrase sichern
 - 5.8) Fixes Kommando ausführen
- 6) Port-Forwarding (Tunneling anderer Protokolle)
- 7) Übersicht
 - 7.1) Kommandos
 - 7.2) Server-Konfigurationsdateien
 - 7.3) Client-Konfigurationsdateien
 - 7.4) 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 diese UNIX/LINUX_Remote-Kommandos ersetzen. Bei den meisten LINUX/UNIX-Versionen ist SSH inzwischen Standard, auf ftp/telnet-Dienste 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 VPN-Funktionalität ("VPN für Arme"). Kurzer Funktionsumfang:

- * Login auf Remote-Host
- * Ausführung von Kommandos auf Remote-Host (interaktiv / nicht-interaktiv)
- * Kopieren von Dateien zwischen Hosts
- * Port-Forwarding/Tunneling
- * Komprimierung

Besondere Eigenschaften:

- * RSA-basierte Authentifizierung (IP-, Routing-, DNS-Spoofing nicht möglich)
- * Systemweite + benutzerbezogene Konfigurationsdateien
- * Übertragung von Binärdaten (+ Komprimierung) möglich
- * Sechs Verfahren der Client-Authentifizierung gegenüber dem Server
- * Automatische und transparente Verschlüsselung der gesamten Kommunikation
- * OpenSSH: Kommt ohne patentierte Algorithmen aus (<http://www.openssh.org>)
- * Arbeitet gut mit anderen Tools zusammen (z.B. "rsync", "X", "cvs", "svn", "unison").
- * Port 22 (SSH-Port) für Kommunikation verwendet
- * Windows-Ersatz: Putty (suchen in Google nach: putty download)

2) Übersicht Konfigurationsdateien

Es gibt zentrale (rechnerspezifische) Konfigurationsdateien im Verz. "/etc/ssh" und benutzerspezifische im Verz. "~/.ssh" jedes Benutzers. Gibt es für den gleichen Zweck beide Varianten, dann überschreiben die benutzerspezifischen Einstellungen die zentralen. Existiert die benutzerspezifische Konfigurationsdatei nicht, dann gilt die zentrale vollständig.

In der Regel gibt es Schlüsselpaare pro Rechner bzw. pro Benutzer, da es 3 verschiedene Verschlüsselungsverfahren gibt (veraltet/modern, unsicher/sicher, patentbehaltet/patentfrei). Die Schlüsselpaare für den Rechner werden bei der Installation des Rechners automatisch angelegt. Die Schlüsselpaare für den Benutzer müssen manuell pro Benutzer per "sshkey-gen" angelegt werden. Ein zusammengehörendes Schlüsselpaar ist in 2 Dateien abgelegt:

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

Lokal (Client)	Kommunikation	Remote (Server)
Client: ssh, scp, sftp	=====> <=====	Server-Dämon: sshd Firewall mit offenem SSH-Port
/etc/ssh/ssh_config ~/.ssh/config	Konfiguration	/etc/ssh/sshd_config (d!)
/etc/ssh/ssh_known_hosts ~/.ssh/known_hosts	<=====	/etc/ssh/ssh_host_rsa_key (P2) /etc/ssh/ssh_host_rsa_key.pub /etc/ssh/ssh_host_dsa_key (P2) /etc/ssh/ssh_host_dsa_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 (P2) ~/.ssh/id_dsa.pub ~/.ssh/identity (P1) ~/.ssh/identity.pub	=====> Client-Authentifiz.	~/.ssh/authorized_keys

HINWEISE:

- * Server-Konfigurationsdatei "sshd_config" (MIT d!) und Client-Konfigurationsdatei "ssh_config" (OHNE d!) NICHT verwechseln!
- * Die Besitzverhältnisse und Zugriffsrechte sämtlicher Konfigurationsdateien und ihrer Verz.pfade müssen einige Bedingungen erfüllen, damit SSH eine schlüsselbasierte Anmeldung erlaubt. Der "Besitzer" von "/home/USER" und "/home/USER/.ssh" und der Datei "authorized_keys" muss identisch zum sich per SSH anmeldenden Benutzer sein. Weiterhin darf das "w"-Recht für "Besitzer-Gruppe" + "Andere" nicht gesetzt sein.
Typische Fehlermeldungen in "/var/log/auth.log" auf der Server-Seite lauten:

```
Authentication refused: bad ownership or modes for directory /home/kurs1
Authentication refused: bad ownership or modes for directory /home/kurs1/.ssh
```

3) Prinzipielle Arten von Verschlüsselung

- a) Symmetrisch: Gleicher Schlüssel zum Ver- und zum Entschlüsseln
 - Schlüsselaustausch muss über geheimen Kanal erfolgen
 - Schlüssel muss geheim gehalten werden
 - + Schnell
- b) Asymmetrisch: 2 verschiedene aber eindeutig zusammengehörende Schlüssel, ein öffentlicher (public key) und ein privater (private key)
 - + Schlüsselaustausch einfach (public key darf JEDER kennen!)
 - + Nur privater Schlüssel muss geheim gehalten werden
 - Langsam
- c) Hybrid: Verbindung der Vorteile beider Verfahren.
Beim Verbindungsaufbau wird über eine asymmetrische Verschlüsselung ein symmetrischer "Sitzungsschlüssel" vereinbart und darüber die eigentliche Kommunikation durchgeführt (bei SSH etwa alle 60 Minuten erneuert).

SSH verwendet hybride Verschlüsselung: Das asymmetrische DSA- oder RSA-Verfahren 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 besorgt würde, nachträglich entschlüsselt werden. Da der Sitzungsschlüssel aber nicht mehr existiert und laufend wechselt, ist eine nachträgliche Entschlüsselung der Sitzung nicht oder nur für kleine Abschnitte der Sitzung möglich ("replay"-Angriff).

4) SSH installieren, aktivieren und testen

 Prüfen, ob SSH installiert ist:

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

Auf dem eigenen Rechner muss der SSH-Client "ssh" 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-Dienst auf dem Remote-Rechner (Server) aktivieren:

```
rcsshd start              # Temporär oder
/etc/init.d/sshd start    # Temporär oder
chkconfig -a sshd        # Permanent (-a=add)
```

Überprüfen ob der SSH-Dämon "sshd" läuft:

```
rcsshd status            # oder
/etc/init.d/sshd status  # oder
ps -aux | grep "sshd"
```

Bei Änderungen an der SSH-Server-Konfigurationsdatei "/etc/ssh/sshd_config" muss SSH-Server neu gestartet und die Verbindungen neu aufgebaut werden:

```
rcsshd restart           # oder
/etc/init.d/sshd restart
```

Änderungen an der SSH-Client-Konfigurationsdatei "/etc/ssh/ssh_config" bzw. "~/.ssh/config" wirken sich automatisch auf danach neu hergestellte Verbindungen aus, nicht aber auf die aktuell laufenden Verbindungen.

Es gibt 2 Protokoll-Versionen von SSH:

```
+-----+-----+
| Version SSHv1: RSA      | P1 |
| Version SSHv2: RSA und DSA | P2 |
+-----+-----+
```

Sie unterscheiden sich in der Aushandlung des symmetrischen Schlüssels, (sicherheitstechnische und patentrechtliche Gründe). Version 1 sollte aus Sicherheitsgründen nicht mehr verwendet werden!

Die wichtigsten Einstellungen von "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                             |
+-----+-----+
```

Neben BENUTZERSPEZIFISCHEN Schlüsseln gibt es auch RECHNERSPEZIFISCHE Schlüssel. Die rechner-spezifischen Schlüssel werden meist 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, von welchen Rechnern überhaupt eine SSH-Verbindung erlaubt ist:

```
+-----+-----+
| /etc/ssh/host_key_rsa      | Version 2 (privater RSA2-Schlüssel) |
| /etc/ssh/host_key_rsa.pub  | Version 2 (öffentlicher RSA2-Schlüssel) |
| /etc/ssh/host_key_dsa     | Version 2 (privater DSA-Schlüssel) |
| /etc/ssh/host_key_dsa.pub  | Version 2 (öffentlicher DSA-Schlüssel) |
| /etc/ssh/host_key         | Version 1 (privater RSA1-Schlüssel) |
| /etc/ssh/host_key.pub     | Version 1 (öffentlicher RSA1-Schlüssel) |
+-----+-----+
```

Testen der SSH-Verbindung:

```
telnet localhost 22      # Antwort z.B. "SSH-1.99-OpenSSH_3.0p1"
```

ESCAPE-Zeichen, um eine SSH-Verbindung abubrechen:

```
~.      # Escape character ist '~'
ESC ]   # Escape character is '^]'
```

Sonstige ESC-Sequenzen zur interaktive Bedienung:

```

~?          # Hilfe anzeigen
~~         # Tilde-Zeichen eintippen
~<Strg-Z>  # SSH stoppen (mit "fg" weiterlaufen lassen)
~&        # SSH in Hintergrund schicken (Tunnel noch genutzt)

```

5) Anwendungen von SSH

5.1) Anmelden per SSH

Per SSH an einem anderen Rechner anmelden (als aktueller Benutzer oder jemand anderer auf dem Remote-Rechner anmelden):

```

ssh 172.23.19.56          # Als aktueller Benutzer
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 172.23.10.56
```

Meldung beim 1. Verbindungsaufbau zu Host, dass Host-Schlüssel unbekannt ist:

```

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)?

```

5.2) Server-Authentifizierung

Wird die Verbindung zu diesem Host mit "yes" (3 Buchstaben!) akzeptiert, dann wird in der lokalen Datei "~/ssh/known_hosts" der rechnerspezifische ÖFFENTLICHE Schlüssel der Gegenstelle eingetragen (mit Zuordnung zu einer IP):

```

/etc/ssh/ssh_host_rsa_key.pub  # Version 2 (öffentlicher Schlüssel)
/etc/ssh/ssh_host_dsa_key.pub  # Version 2 (öffentlicher Schlüssel)
/etc/ssh/ssh_host_key.pub     # Version 1 (öffentlicher Schlüssel)

```

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

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

Das ist bequem aber evtl. GEFÄHRLICH, da der Schlüssel kompromittiert sein könnte ("man-in-the-middle"). Daher sollte er über einen getrennten Kanal transportiert werden (z.B. USB-Stick, Diskette, CDROM). Zumindest sollte der "Fingerprint" (16 Byte lange Hexadezimalzahl) aus dem übertragenen Schlüssel gebildet und mit dem des Originals verglichen werden:

```

ssh-keygen -l                  # -l=list (Standarddatei vorgeschlagen)
ssh-keygen -l -f SCHLÜSSELDATEI # -l=list, -f=file

```

Wird von SSH bei der Erstellung eines neuen Schlüssels bzw. bei der ersten Verbindungsaufnahme angezeigt (z.B. am Telefon vorlesen lassen oder von der Visitenkarte ablesen).

```
RSA key fingerprint is 8a:58:19:a1:27:95:2b:e4:e2:3e:5f:41:c7:58:2a:d2.
```

Übung

* Oben mit "yes" akzeptieren -> in "~/ssh/known_hosts" steht anschließend IP-Adresse + Schlüssel + Info der Gegenstelle.

* Schlüssel in "known_hosts" verändern (z.B. ein Zeichen übertippen)
-> Anmeldung geht nicht mehr

* Andere IP-Adresse eintragen (steht in "known-hosts" mit drin) -> Analog.
Es ergibt sich in beiden Fällen folgende Fehlermeldung:

```

@@@@@@@@@@@@@@@@@@@@@...
@ WARNING...
@@@@@@@@@@@@@@@@@@@@@...
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
...

```

Dies ist ein häufiger Fehler, wenn man SSH einsetzt. Lösungsmöglichkeiten:

- * Eintrag aus "~/.ssh/known_hosts" entfernen
- * Eintrag aus "/etc/ssh/ssh_known_hosts" entfernen
- * "StrictHostKeyChecking" auf "no" setzen

5.3) Kommando per SSH remote ausführen

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

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

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

```
ssh HOST ls -l *      # "*" auf lokalem Rechner ausgewertet
ssh HOST ls -l \*    # "*" auf Remote-Rechner ausgewertet
ssh HOST ls -l "*"    # "*" auf Remote-Rechner ausgewertet
ssh HOST ls -l '*'   # "*" auf Remote-Rechner ausgewertet
ssh HOST "ls -l *"   # "*" auf Remote-Rechner ausgewertet
ssh HOST 'ls -l *'   # "*" auf Remote-Rechner ausgewertet
```

5.4) Dateien per SSH zwischen Rechnern kopieren

- * ACHTUNG: ":" nicht vergessen, sonst erhält man eine lokale Kopie!
- * Standard-Benutzer ist aktuell angemeldete Benutzer
- * Standard-Verz. ist sein Heimat-Verz.
- * Passwort wird verlangt, wenn kein Schlüsselaustausch stattgefunden hat
- * Struktur des Filesystems auf dem anderen Rechner muss bekannt sein, damit man dort mit korrekten Pfaden zugreifen kann
- * Zugriffsrechte auf anderem 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: Als anderer Benutzer anmelden ("-l" geht nicht!)
scp HOST:FILE .     # Remote=Quelle: "FILE" im Heimat-Verz. in akt. Verz.
scp USER1@HOST1:FILE USER2@HOST2: # Zw. zwei fremden Rechnern kopieren
```

Einige wichtige Optionen von "scp":

-C	Komprimierung
-l LIMIT	Max. Bandbreite in Kbit/s (limit)
-o OPTION=VALUE,...	SSH-Option
-p	Zugriffsrechte + zeiten erhalten (preserve)
-P NR	Port NR statt 22 (GROSS wg. -p=preserve!)
-q	Weniger Meldungen (Fortschrittanz.; quiet)
-r	Rekursiv (alle Verz. + ihre Inhalte)
-v	Mehr Meldungen (verbose)

5.5) X Window-Protokoll tunneln

X Window-Protokoll durchtunneln (Variable "DISPLAY" anschließend gesetzt):

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

Um X Window-Anwendungen remote starten zu können, 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) / "-x" (disable) bzw. "-Y" (enable trusted) beim ssh-Aufruf mitgeben).
- * Evtl. "xhost +HOST" auf lokalem Rechner (macht eigentlich "xauth")

Öffentlichen Schlüssel vom fremden 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 HOST       # V1: RSA-Schlüssel von HOST holen
```

5.6) Client-Authentifizierung (ohne 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 Anwender ein Schlüssel generiert werden, dessen öffentlicher Teil auf dem Remote-Rechner (Server) in eine Datei "authorized_keys" einzutragen ist.

Zunächst ist zu einem Benutzer kein Verz. "~/.ssh/" in seinem Heimat-Verz. vorhanden. Dieses wird erzeugt, wenn für ihn ein Schlüsselpaar generiert wird (ohne "Passphrase" = leere Passphrase!):

```
ssh-keygen -t rsa          # V2: Patentiert bis Mitte 2004 (-t=type)
ssh-keygen -t dsa         # V2
ssh-keygen -t rsa1       # V1
```

Man erhält einen "Fingerprint" (16-stellige Hexadezimalzahl) des öffentlichen Schlüssels ausgegeben, den man z.B. auf seiner Web-Seite veröffentlicht oder in seine Visitenkarte reindruckt. Andere können darüber manuell feststellen, ob der Verbindungspartner wirklich der richtige ist.

Der Weg, auf dem der öffentliche Schlüssel ausgetauscht wird, darf nicht manipulierbar sein (z.B. per Hand, Telefon, Diskette, USB, Maus per SSH-Verbindung, NICHT per eMail!). Eine weitere Möglichkeiten wäre der Transport des Schlüssels mit der Maus in einer SSH-Anmeldung

```
cat ~/.ssh/id_rsa.pub
ssh USER@HOST
mkdir ~/.ssh
vi ~/.ssh/authorized_keys
... angezeigten Schlüssel mit Maus als 1 Zeile einfügen ...
:wq
exit
```

Oder mit "scp":

```
scp ~/.ssh/id_rsa.pub USER@HOST:/tmp
ssh USER@HOST
mkdir ~/.ssh
cat /tmp/id_rsa.pub >> ~/.ssh/authorized_keys
exit
```

Oder alles auf einen Schlag:

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

ACHTUNG: Kein Anfangs/Endstück weglassen, muss EINE Zeile sein (keine Zeilenumbrüche).

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

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

USER1	ANMELDUNG	USER2
ssh USER2@HOST	=====>	ssh
Client (local)	SSH-Protokoll	Server (HOST) (remote)
(per Hand ssh-keygen -t rsa) /home/USER1/.ssh/id_rsa (NICHT KOP!) /home/USER1/.ssh/id_rsa.pub		(beim Installieren erzeugt) /etc/ssh/ssh_host_rsa_key (NICHT KOPIEREN!) /etc/ssh/ssh_host_rsa_key.pub
	(meist automatisch bei 1. ssh + "yes")	
+-----+ v	+-----+ v	
/home/USER1/.ssh/known_hosts # oder /etc/ssh/ssh_known_hosts	(manuell pro Benutzer)	/home/USER2/.ssh/authorized_keys (NICHT: /etc/ssh/authorized_keys!)

Zusammensammeln der bei allen Benutzern (inkl. "root") stehenden "known_hosts" in der zentralen "ssh_known_hosts" (doppelte Einträge nur 1x). Ob das weise ist, sei dahingelassen. Die zentralen Schlüssel in "ssh_known_hosts" sollten VERIFIZIERT sein, die benutzerspezifischen Schlüssel könnten durch Akzeptieren mit "yes" bei der ersten Verbindungsaufnahme erzeugt sein und sind dann NICHT

verifiziert.

```
cat /home/*/.ssh/known_hosts > /tmp/knu          # Umlenken + sudo nicht glz.
sudo cp /root/known_hosts /tmp/knr
cat /tmp/knu /tmp/knr | sort | uniq > /tmp/kns    # Umlenken + sudo nicht glz.
sudo cp /tmp/kns /etc/ssh/ssh_known_hosts
```

NACH Austausch der öffentlichen Schlüssel wird häufig in "/etc/ssh/sshd_config" die Authentifizierung via Passwort ausgeschaltet:

RSAAuthentication	yes	Reine RSA-Authentifizierung erlaubt
PubkeyAuthentication	yes	Public-Key Authentifizierung erlaubt (nur P2)
PasswordAuthentication	no	SSH fragt selber nach dem Passwort
UsePAM	no	Sonst wird Passwort von PAM verlangt

Wichtig sind weiterhin folgenden Einstellungen in "/etc/ssh/sshd_config":

PermitRootLogin	no	Direkte root-SSH-Anmeldung nicht möglich
PermitEmptyPasswords	no	Leere Passworte verboten
IgnoreRhosts	yes	Datei "~/.rhosts" "~/.shosts" nicht benutzen
IgnoreUserKnownHosts	yes	Userspez. Hosts "~/.ssh/known_hosts" verboten

Weitere sinnvolle Einstellungen in "/etc/ssh/sshd_config" sind:

StrictModes	yes	Zugriffsrechte auf Dateien prüfen
Port	55022	Port 22 -> 55022 (gegen Bot-Attacken)
Protocol	2	Protokoll 1 ist unsicher
ChallengeResponseAuthentication	no	
TCPKeepAlive	yes	TCP-Keepalive Meldungen schicken
X11Forwarding	yes	X11-Protokoll tunneln
X11DisplayOffset	10	X11-Display Startnummer

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 auf der Platte liegt (Diebstahl oder Kompromittierung des eigenen Rechners führt dann nicht zur Kompromittierung des privaten Schlüssels). Dann wird beim Remote-Login per SSH nach der Passphrase des Schlüssels gefragt (damit dieser entschlüsselt werden kann), nicht mehr nach dem Passwort.

```
$ ssh-keygen -p -f ~/.ssh/id_dsa      # f=file
$ ssh-keygen -p                      # 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): XXXXXXXXXX.
Enter same passphrase again: XXXXXXXXXX
Your identification has been saved with the new passphrase.
```

Diese "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). Alternativ kann die Passphrase über einen SSH-Agent zur Verfügung gestellt werden, sodass sie nur lx eingegeben werden muss, im Speicher liegt und dann bei jedem SSH-Kommando automatisch für den Zugriff verwendet wird:

```
eval $(ssh-agent -s)                # -s Bourne-Shell Befehle erzeugen
ssh-add < /dev/null                 # Passphrase lx eingeben (per grafischem Eingabedialog)
ssh-add                             # Passphrase lx eingeben (auf der Kommandozeile)
```

Am besten dafür einen Alias "sshadd" definieren, um sich nicht soviel kompliziertes Zeug merken zu müssen:

```
alias sshadd="eval $(ssh-agent -s); ssh-add"
```

5.8) Fixes Kommando ausführen

In "authorized_keys" VOR dem Schlüssel-Typ die Option "command=..." angeben (alles in einer Zeile, das Kommando in "..." oder '...' einschließen

```
command="/bin/ls > /tmp/ls.txt"      ssh-rsa ...
```

```
command="/usr/bin/ssh tom@192.168.1.1" ssh-rsa ...
```

Die Shell `"/bin/bash"` muss eingetragen sein, damit das Kommando ausgeführt werden kann. Der Eintrag in `"authorized_keys"` sorgt dafür, dass keine echte Anmeldung mehr erfolgt, sondern das in `"..."` angegebene Kommando ausgeführt und danach die SSH-Verbindung wieder beendet wird (sofern die schlüsselbasierte Anmeldung mit diesem Schlüssel stattfindet). Bei einer passwortbasierten Anmeldung wird kein Kommando ausgeführt). Sinn:

- * Einzelne Kommandos freischalten
- * SSH-Verbindung nach außen über Proxy-Server (keine End-zu-End-Verbindung)
- * SSH-Datenfluss überwachen

6) Port-Forwarding (Tunneling anderer Protokolle)

Durchschleusen von anderen Protokollen durch einen SSH-Tunnel. Insbesondere Klartext-Protokolle (z.B. TELNET, FTP, POP3, IMAP, SNMP) können auf diese Weise sicher übertragen werden. Auf dem Zielrechner "SERVER" muss man sich per SSH anmelden können und folgende Einstellung muss dort gelten:

```
AllowTcpForwarding yes
```

- A) Von Port 8080 lokal auf Port 80 auf dem remote SERVER tunneln (keiner kann 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" # mit Passwortabfrage
Strg-Z
bg
```

Im Browser den lokalen Rechner auf Port 5000 ansprechen, die HTTP-Daten werden verschlüsselt an/vom SERVER auf Port 80 übertragen:

```
http://localhost:8080
```

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

```
ssh -L 5000:localhost:110 SERVER
fetchmail (eine gültige ".fetchmailrc" muss existieren,
die auf localhost:5000 zugreift)
```

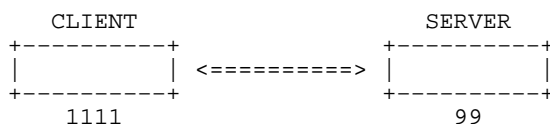
- C) Verschlüsselte Telnet-Verbindung lokal auf Port 5000 (Port 23) zum Rechner "SERVER" aufbauen (vorher telnet-SERVER installieren und aktivieren sowie "xinetd" rekonfigurieren und neu starten):

```
ssh -R 5000:CLIENT:23 SERVER "sleep 100000"
ssh -p 5000 CLIENT
```

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

```
ssh -L 3306:localhost:3306 SERVER
```

Allgemein gibt es folgende Möglichkeiten des "Tunneling" gesteuert durch ein SSH-Kommando auf dem Client:



Verbindung vom CLIENT zum SERVER aufmachen + lokal lauschen auf 1111, weitergeben an SERVER auf Port 99:

```
CLIENT: ssh -L 1111:localhost:99 SERVER # -L (local)
CLIENT: telnet localhost 1111 # Tunnel zu SERVER:99
```

Verbindung vom SERVER zum CLIENT aufmachen + remote lauschen auf 1111, weitergeben an CLIENT auf Port 99 (CLIENT statt "localhost" nach `-R ...!`):

```
CLIENT: ssh -R 99:CLIENT:1111 SERVER # -R (remote)
SERVER: telnet localhost 99 # Tunnel zu CLIENT:1111
```

Verbindung von "localhost" über HOST2 (SSH-Port 4021) zu HOST3 aufbauen (2 Tunnels!) und das HTTP-Protokoll (Port 80) von HOST3 zu Port 10001 lokal tunneln:

```
ssh -l root -p 4021 -L 10001:localhost:3000 HOST2 \
"ssh -L 3000:localhost:80 HOST3"
```

Ablauf: Anmelden auf Port 4021 am Rechner HOST2
 Tunneln von Port 10001 auf eigenem Rechner LOCALHOST
 zu Port 3000 auf mittlerem Rechner HOST2 (-L = "localhost" = dort)
 zu Port 80 auf Endrechner HOST3 (-L = "localhost" = dort)

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 einem Rechner holen
ssh-add	Privaten Schlüssel beim "ssh-agent" registrieren
ssh-agent	Privaten Schlüssel verwalten (automatische Beantwortung von "challenges")

7.2) Server-Konfigurationsdateien

/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_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 Server (fest)

7.3) Client-Konfigurationsdateien

/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/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 Server (updatebar)
~/.ssh/authorized_keys	Public Keys der bekannten Benutzer

7.4) Links

* http://www.ssh.fi	SSH (original)
* http://www.openssh.com	OpenSSH
* ftp://ftp.de.openbsd.org/pub/unix/OpenBSD/OpenSSH/portable	OpenSSH
* http://www.chiark.greenend.org.uk/~sgtatham/putty	Putty (Windows SSH)
* ftp://ftp.freeware.com/pub/infozip/zlib	Zlib
* http://www.openbsd.org	OpenBSD
* http://www.openssl.org	OpenSSL