Installation
Anwendung
Interne Mechanismen

Hinweis: Diese Kursmaterialen zur Secure Shell sind nicht mehr uptodate und demzufolge nicht anwendbar!

Kursmaterialien ab dieser Seite obsolet!!

- Einleitung
- Hintergrund
- Vorteile gegenüber bisherigen Mechanismen
- Benutzung von ssh und scp

- Komponenten des Softwarepaketes
- Installation: Administrator und User
- Authentisierung: Einloggen und Filetransfer ohne Passwort
- Benutzung unter Windows

- Tunneling
- Interna: Protokolle, Verschlüsselung
- Lizensierung

Ansprechpartner

Hans Schummer ---------- hans.schummer@rhrk.uni-kl.de
Joachim Backes ---------- joachim.backes@rhrk.uni-kl.de
Dr. Tonnis Pool ---------- tonnis.pool@rhrk.uni-kl.de

Einleitung

- SSH in verschiedenen Varianten am RHRK verfügbar
- LINUX und AIX: OpenSSH
- Windows: z.B. Putty
- Ausschluss unberechtigter Zugriffe
- Verschlüsselter Zugang

Verfügbare Plattformen

- UNIX mit LINUX
- Windows

- http://www.ssh.com
- http://www.openssh.org
- http://www.freessh.org

Internet-Protokolle weisen z.T. schwerwiegende Sicherheitsprobleme auf

- Unverschlüsselte Datenübertragung
- Aufzeichnen fremder Kommunikation durch Sniffer (ethereal) für Passwörter, Benutzernamen
- Fehlende Sicherung von Authentizität und Integrität
- Verfälschung der Identität möglich
- IP-Spoofing: Senden gefälschter Pakete

- Server die Authentisierung mittels IP-Adresse vornehmen, angreifbar
- Vorgaukeln eines trusted hosts
- Gefährdet: telnet
- Gefährdet: rsh, rlogin, denn ~/.rhosts und /etc/hosts.equiv sind Angriffspunkte

Vorteile der SSH (Secure Shell)

- SSH garantiert, dass sich ein Host nicht als ein anderer ausgibt
- Keine Probleme mit Routing und ~/.rhosts bei Rechnern mit mehreren Interfacen
- Informationsübertragung verschlüsselt während Verbindungsaufbau und Datenphase

- Geheimhaltung der Schlüssel durch
  - Asymmetrisches Public-Key-Verfahren
  - Später symmetrische Verschlüsselung
- Fakultativ: komprimierte Datenübertragung

- Schlagwort Tunneling:
- Port-Forwarding erlaubt Verbindungen
  - (TCP/IP) oder Zugriffe auf Remote-Ports
  - (UDP) über einen verschlüsselten Kanal.
- Beispiele: POP3 oder FTP

- Sicherheit bei X11-Anwendungen
- SSH-Sessionaufbau innerhalb einer X11-Konsole zu einem Remote-Host
- Automatisches Setzen von $DISPLAY
- Dort Start einer X11-Anwendung: Übertragung zum X-Server verschlüsselt

- Xhost +... oder zusammen mit ~/.Xauthority können entfallen
- Aber: gebremste Performance bei aufwendigen Grafiken

Benutzung von ssh und scp

- ssh: Aufbau einer Sitzung zu einem fremden Rechner
- Absetzen von Kommandos auf einem fremden Rechner
- Kompletter Ersatz für
  - telnet
  - rsh
  - rlogin

Benutzung von ssh und scp (UNIX/Linux)

- Vereinfachter Aufruf von ssh:
  - ssh [remoteuser]@remotehost [kommando] oder
  - ssh -l [remoteuser] remotehost [kommando]

- Mit Schalter '-v': erweiterte Protokollierung für Fehlersuche

- Vereinfachter Aufruf von scp:
  scp [-v] sourcefile destinationfile
- Layout der Fileangaben sourcefile und destinationfile:
  [[user]@host:]path

- scp bietet ähnliche Funktionalität wie das Kommando rcp
- Zusätzlich aber: Passwortabfrage bei Fehlen entsprechender Voraussetzungen

- Bei erstmaligem Zugriff auf den Remotehost: Frage, ob Verbindung aufzubauen
- Entfällt bei späteren Zugriffen
- Aber: erfolgt wieder bei Schlüsseländerung des Remotehosts (oder bei anderen Problemen mit dem Remotehost)
- Folge: in einem Batch- oder Cronjob ist es nicht möglich, einen anderen Partner untergeschoben zu bekommen.

- Sensitiv gegenüber Berechtigungen im .ssh-Directory (Option: -v !)
- Immer gilt im Zweifelsfall: es erfolgt Passwortabfrage

- Als Systemadministrator: Installation der Client- und Serverkomponenten
- Als nichtprivilegierter Anwender: Installation der Client-Komponenten

- Client-Schnittstellen (bei Installation in **/usr/local/bin**):

  /usr/local/bin/scp
  /usr/local/bin/slogin
  /usr/local/bin/ssh
  /usr/local/bin/sshadd
  /usr/local/bin/sshagent
  /usr/local/bin/sshaskpass
  /usr/local/bin/sshkeygen
  /usr/local/bin/sftp

- SSH-Daeomon:

  /usr/local/bin/sshd

Software-Komponenten

- Es gibt zwei Protokollvarianten der ssh-Software, bedingt durch Lizenzprobleme mit den diversen Verschlüsselungsalgorithmen:
  - ssh1 (obsolet)
  - ssh2

Konfigurationsdateien SSH (ssh2, aktuell!)

/etc/ssh/ssh_config
/etc/ssh/sshd_config

/etc/ssh/ssh_host_key.pub — Der öffentliche Rechner-Hostkey
/etc/ssh/ssh_host_key — Der private Rechner-Hostkey

Reine Benutzerinstallation

Vom Benutzer aufgerufene Binaries sind in beliebigen Directories aufrufbar:

- ssh
- scp
- ssk-keygen

Einloggen ohne Passwortabfrage (UNIX/LINUX)

- Auf dem lokalen Host mit ssh-keygen -P -b 1024 ein 1024 Bit langes Schlüsselpaar für den geheimen und öffentlichen Schlüssel generieren
- Wird automatisch abgelegt in ~/.ssh unter den Namen
  - id_rsa.<...> und
  - id_rsa.<...>.pub

```
erich@localsys $ ssh-keygen -P -b 1024  # Leere
Passphrase durch -P
Generating 1024-bit dsa key pair
6 Oo.oOo.ooOo.
Key generated.
1024-bit dsa, erich@localsys.rhrk.uni-kl.de, Fri Sep 13
2002 10:57:22 +0200
Private key saved to
/usr/people/erich/.ssh2/id_dsa_1024_b
Public key saved to
/usr/people/erich/.ssh2/id_dsa_1024_b.pub
erich@localsys $
```

- Passphrase: Funktion eines zweiten Passwortes, schützt den privaten Schlüssel
- Dient der Erzeugung des Schlüsselpaares
- Wird u.U. beim Verbindungsaufbau erfragt
- Einloggen ohne Passwort gewünscht: Leerer String (Zeilenende) für Passphrase, hier durch die Option -P erreicht
- Ansonsten: immer Passphrase verwenden

Problematisch: ssh-Kommandos innerhalb Batch- oder Cronjobs, mit Passphrase

```
erich@localsys $ ssh erich@remotesys
Host key not found from database.
Key fingerprint:
frtdh-cakel-cesim-gprxh-nukag-dkrin-nomem-fuget-digir-kakoz-fyxax
You can get a public key's fingerprint by running
% ssh-keygen -F publickey.pub on the keyfile.
Are you sure you want to continue connecting (yes/no)? yes
Host key saved to /usr/people/erich/.ssh/hostkeys/key_22_remotesys.pub
host key for remotesys, accepted by erich Fri Sep 13 2002 11:19:47 +0200
erich's password: ........
Authentication successful.
Last login: Fri Sep 13 2002 11:17:18 +0200 from localsys.rhrk.uni-kl.de
no mail.
erich@remotesys $
```

- Zielsystem OK?
  Test beispielsweise durch Kommando ls -l
- Bei Änderung des privaten Schlüssels des Remotehosts:
  SSH reagiert sofort mit der Nachfrage
  Are you sure you want to continue connecting (yes/no)?
- Problem wiederum: bei Batch- oder Cronjobs

Nächste Schritte im Remotehost:

Anlegen der .ssh-Hierarchie
erich@*remotesys* $ cd
erich@*remotesys* $ ssh-keygen2 -P -b 1024 # wie auf localhost
erich@*remotesys* $ echo Key localsys.pub >> ~/.ssh/authorization

~/.ssh/authorization: Datei, die die Namen der Dateien enthält, in denen die öffentlichen Schlüssel der Systeme localsys aufgeführt sind.

- Letzter Schritt im lokalen Host *localsys*:
  Kopie des öffentlichen Schlüssels auf den Remote-Host remotesys
  erich@*localsys $* cat $HOME/.ssh/id_rsa.<...>.pub |

      ssh -x erich@remotesys "cat >.ssh/<...>.pub"
      erich's password:....

- Nun einloggen auf dem Remote-Host remotesys ohne Passwort:
  erich@localsys $ ssh remotesys
  erich@remotesys $

Zu aufwendig? Verwende das Shellskript

/usr/rhrk/make_ssh_entry

auf linda.rhrk.uni-kl.de oder lindb.rhrk.uni-kl.de

```
#!/bin/bash
#################################################################
#
# Make ssh entry for an ssh.com computer to an openssh computer
#
# Backes, September 13, 2002
#
#################################################################

cd ~/.ssh
trap "stty echo" EXIT
[ -f id_dsa.pub ] || {
  echo Generating local key ...                 # local
  /usr/bin/ssh-keygen -b 1024 -t dsa            # public key
  echo Generating local key done.               # erzeugen
}

while true
do
   echo -n "Name des Zielrechners, oder Ende mit Ctrl-C [$HOST]:"
   read HOST1
   [ "$HOST1" != "" ] && HOST=$HOST1
   [ "$HOST" = "" ] && continue

   ping -c 1 $HOST 2>/dev/null >/dev/null || { echo Bad host name: $HOST; continue; }

   echo -n "User-Name fuer Rechner $HOST [$USER]:"
   read USER1
   [ "$USER1" != ""  ] && USER=$USER1

   echo
   /usr/bin/scp -q id_dsa.pub $USER@$HOST:xxyyzz        # kopiere public key zum Ziel
   HOSTNAME=`hostname`                                  # in Hilfsdatei
   /usr/bin/ssh -x -t -q $USER@$HOST "
        chmod 700 ~
        [ -d .ssh ] || mkdir .ssh
        cat xxyyzz  >> .ssh/authorized_keys2            # public key anfuegen
        chmod 700 .ssh/authorized_keys2
        rm -f xxyyzz                                    # loesche hilfsfile
   "
   echo Achtung: Ab jetzt darf keine Passwortanfrage erscheinen, sonst war der Eintrag nicht korrekt!!
   /usr/bin/ssh  -qx $USER@$HOST "echo -e \\\nEintrag auf $HOST erfolgreich\\\n"

   done
```

- Skript fragt in einer Schleife nach den Hosts, zu denen der Zugang ohne Passwort freigeschaltet werden soll
- Skript erstellt die erforderlichen Einträge
- Gegebenenfalls Anpassung an das eigene System erforderlich (Kommando ping, ssh-Installationsort,...)

Verwendung von ssh-agent mit XDM

- Bei Aufbau einer ssh-Session innerhalb eines Terminal-fensters werden Remote-Host-Ausgaben verschlüsselt übertragen
- Performance-Problem, mit der ssh-Option -x abschalt-bar.
- Aber: Was tun beim Session-Aufbau mit XDM (solange es noch kein Terminalfenster gibt!)?
- Lösung: Verwendung von ssh-agent/ssh-add

Aktionen auf dem Remote-Host remotesys

- Ein Schlüsselpaar mit ssh-keygen erzeugen, nicht mit leerer Passphrase (ohne -P!) verwenden
- ~/.xsession nach ~/.xsession-ssh kopieren.
- Neue ~/.xsession erzeugen mit folgendem Inhalt:

```
#!/bin/sh
ssh-agent $HOME/.xsession-ssh # Start von ssh-agent
```

- ln -s ~/.xsession ~/.xinitrc
  (damit XDM-Verbindungsaufbau funktioniert)

- In ~/.xsession-ssh möglichst weit vorne das Kommando

  ssh-add </dev/null

  eintragen. Dieses Kommando fragt einmalig nach der Passphrase in einem kleinen X-Fenster.

- Kopiere lokale Datei ~/.ssh2/id_dsa_1024_a.pub (lokaler öffentlicher Schlüssel) auf alle Rechner, die Ausgaben auf dem lokalen Display erzeugen sollen:

```
cat ~/.ssh2/id_dsa_1024_a.pub |
ssh erich@remotesys "
    mkdir -p .ssh2
    cat >.ssh2/localsys.pub
    chmod 644 .ssh2/localsys.pub"
```

Was spielt sich im einzelnen ab?

- Einloggen über XDM
- ~/.xsession wird ausgeführt
- ~/.xsession startet ssh-agent
- ssh-agent startet ~/.xsession-ssh
- dort fragt ssh-add nach Passphrase
- Passphrase wird von ssh-agent verwendet und wird nicht jedesmal neu angefragt

Installation der SSH unter UNIX

- Von http://openssh.org/portable.html#ftp die Sourcen herunterladen.
- Aktuell: openssh-5.7p1.tar.gz

- Standard-Kommandofolge zur Installation der SSH
  - **cd downloadverzeichnis**
  - **gtar zxf openssh-5.7p1.tar.gz**
  - **cd openssh-5.7p1**
  - **./configure**
  - **make**
  - **sudo make install**

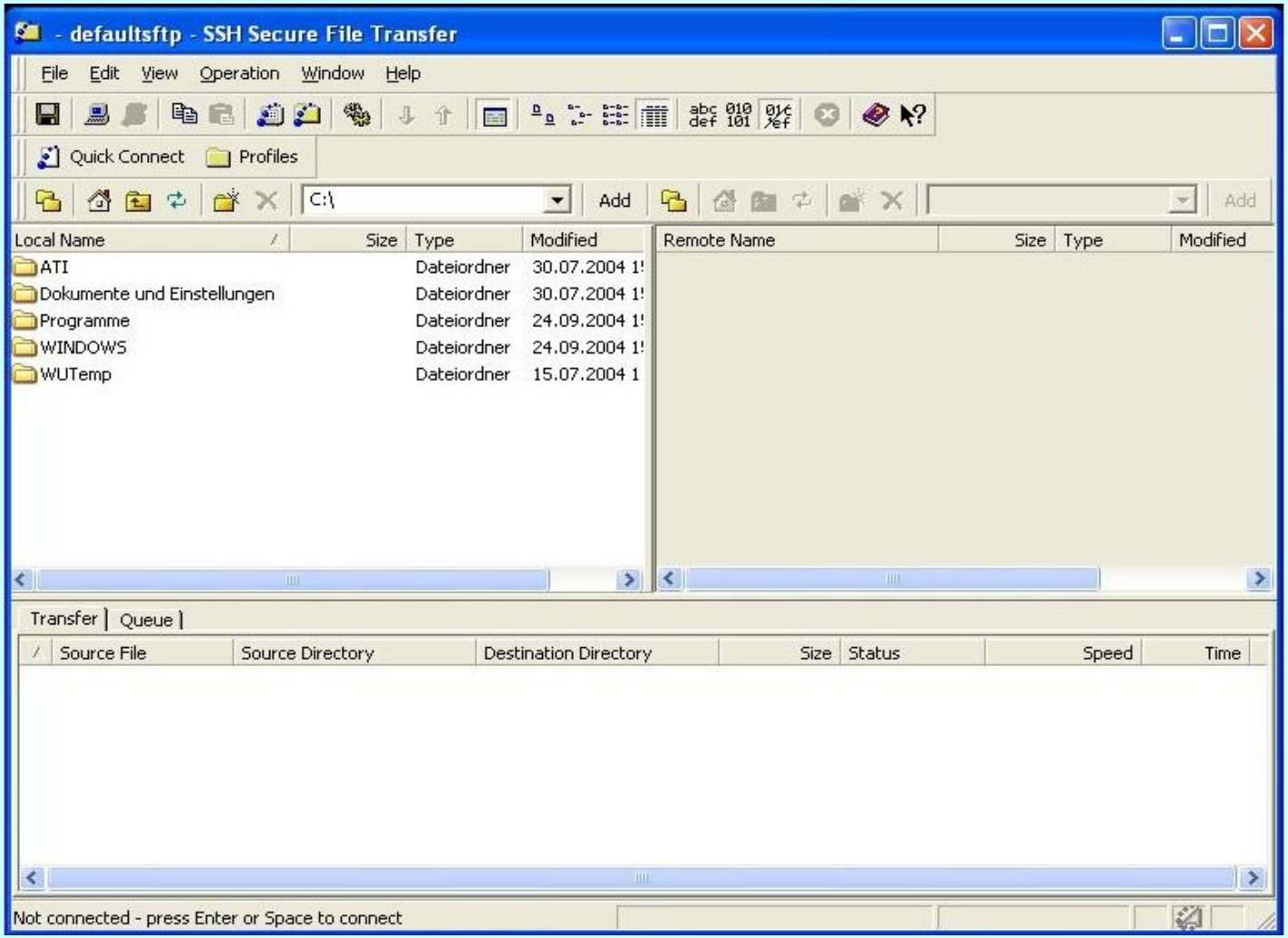
Windows-Implementierungen

Es gibt diverse freie Implementierungen der SSH für Windows . Download z.B. über

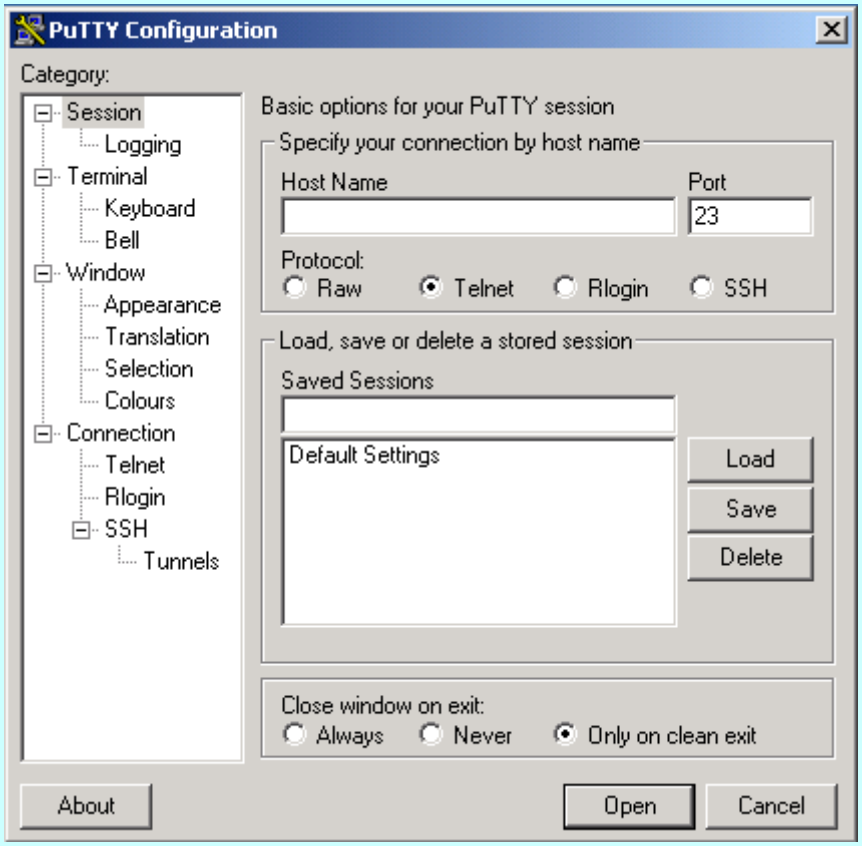
http://www.shuttle.de/support/tutorial/ssh.html

- SSHSecureShellClient-3.2.9.exe
- putty.exe

Beispiel: SSH Secure File Transfer per SSHSecureShellClient



Beispiel: putty.exe



Dokumentation in Form von HTML-Dokumenten für SecureShell.exe

- Beschreibungen
  http://www.fz-juelich.de/zam/docs/tki/tki_html/t0348/tki-0348.htm

- Benutzerhandbuch
  http://www.fz-juelich.de/zam/docs/bhb/bhb_html/d0160/

- Speziell FTP-Tunneling unter Windows
  http://www.fz-juelich.de/zam/docs/bhb/bhb_html/d0160/kap08.htm

Tunneling unter UNIX

Tunneling: Verwendung von SSH zum Durchschleusen a-priori nicht ssh-fähiger Protokolle

- Wie geht man z.B. unter UNIX (für FTP) vor?
- Beispiel: FTP nicht ssh-fähig

- Starte ein separates Terminal auf Localhost
- Baue in diesem Fenster eine ssh-Verbindung zu Localhost auf:
  ssh -R *<lport>:<remotehost>*:21 localhost
- *lport*: beliebige nichtprivilegierte Portnummer > 1024
- 21: Standard-FTP-Port
- ssh kreiert *lport* auf Localhost
- ssh-Daemon (sshd) kreiert intern einen Port *hport* auf Remotehost
- User startet ftp zum Localhost auf Port *lport*:
  ftp localhost <lport>
- SSH leitet Transport verschlüsselt über *lport* zum Remotehost/sshd auf Port *hport*
- sshd auf Remotehost baut Verbindung über inetd für Port 21 auf

Interna der Verschlüsselung

Symmetrische Verschlüsselung

- Sender und Empfänger einigen sich über einen gemeinsamen Key
- Folge: Sicherheit hängt vom Bekanntheitsgrad des Keys ab
- Krypto-Algorithmus darf offengelegt sein
- Brute-Force-Angriffe durch Einmal-Keys aus sequentiell verarbeiteter Liste vermeidbar

- Algorithmus: DES (Digital Encryption Standard)
  Keylänge: 56 Bit
- Verbesserung: 3DES (Triple DES)
  Dreifache Anwendung von DES:
  normal, invertiert, normal
- Abgelöst durch AES (Advanced Encryption Standard)
- Weitere Verfahren:
  - BlowFish
  - IDEA (128 Bit Keylänge)
  - CAST128

Asymmetrische Verschlüsselung

- Jeder Kommunikationspartner besitzt Key-Paar
  - Public Key (an Partner verteilt)
  - Private Key (geheim)
- Aufwand: Key-Erzeugung und -Verteilung, Holen der öffentlichen Keys der Partner
- Vorteil: Aufbau sicherer Kommunikation über unsichere Kanäle
- Nachteil: eher langsam
- Kompromiss: Asymetrische Verschlüsselung zur Erzeugung von Session-Keys, danach symmetrische Verschlüsselung

Interner Ablauf beim Verbindungsaufbau

Vor der Bearbeitung von Verbindungs-Wünschen durch SSH-Clients sind u.a. folgende Schritte erforderlich:

- Auf dem Server-Rechner wird vom Administrator einmalig bzw. in größeren Zeitabständen (mindestens mehrere Monate) ein Host-Key-Paar (RSA) generiert. Für diesen Verwendungszweck sind die Keys i.a. 1024 Bit lang.

- Der laufende Server-Prozess erzeugt periodisch (i.a. Jede Stunde) zusätzlich ein Server-Key-Paar (RSA), das nur im Memory gehalten und nach Ablauf der Zeitspanne gelöscht wird. Diese Keys sind i.a 768 Bit lang.

Aus: www.lrz-muenchen.de/services/security/ssh/ssh-4.html

Bei jeder neuen Verbindung laufen am Anfang folgende Teilschritte ab:

- Der SSH-Client ssh (scp setzt intern ebenfalls auf ssh auf) wendet sich an den SSH-Server auf einem Remote-Rechner.

- Der SSH-Server schickt sowohl seinen Host- als auch seinen Server-Public-Key (siehe oben) an den Client.

- Der SSH-Client kann nun überprüfen, ob er den Server kennt, indem er in Tabellen von bekannten Public-Keys nach dem gerade erhaltenen Host-Public-Key sucht.
- Ist dieser Key noch nicht bekannt, teilt der Client dies dem Benutzer mit und fragt nach, ob die Verbindung trotz der Unsicherheit (die Identität des Servers kann ja nicht überprüft werden) aufgebaut werden soll.

- Bei einer positiven Antwort trägt der Client den Server-Public-Key außerdem in eine individuelle Liste des Benutzers ein; bei der nächstenVerbindung ist dann der Remote-Rechner schon bekannt, und der Client muss nicht mehr nachfragen.

- Bei einer negativen Antwort des Benutzers hört der Client an dieser Stelle wie gewünscht auf.

- Der Client schickt dem Server eine 256 Bit lange Zufallszahl, die sowohl mit dem Host- als auch mit dem Server-Public-Key des Remote-Rechners verschlüsselt wurde.
- Ab sofort werden alle Daten der Verbindung verschlüsselt. Dabei dient die im vorherigen Schritt erzeugte Zufallszahl (bzw. je nach Algorithmus auch nur Teile von ihr) als Sitzungs-Key. Dieser Sitzungs-Key ist bei jeder Verbindung anders, da er jedesmal individuell und zufällig erzeugt wird.

- Der Schutz der Verbindung ergibt sich durch folgende Punkte:
- Selbst wenn der Remote-Rechner eine falsche Identität vorgespiegelt hat, kann dem Benutzer nichts passieren: Da die Zufallszahl mit dem Host-Public-Key des richtigen Rechners verschlüsselt wurde, kann auch nur der richtige Rechner diese Zahl wieder entschlüsseln. Ein falscher Rechner kann dies nicht und würde deshalb für den Rest der Verbindung nur Datenmüll erhalten, der für den Angreifer vollkommen wertlos ist.

- Die zusätzliche Verschlüsselung der Zufallszahl mit dem Server-Public-Key hat folgende Funktion: Selbst wenn ein Hacker in den Remote-Rechner eindringt, hat er nur Zugriff auf Verbindungen, die mit dem aktuell gültigen Server-Public-Key gestartet wurden; deswegen wird der Server-Public-Key periodisch gewechselt.

- Client und Server führen einen Dialog, durch den der Server die Identität und Berechtigung des Clients überprüfen kann. Ist der Client unbekannt oder nicht berechtigt, wird die Verbindung an dieser Stelle abgebrochen.
- In einem weiteren Dialog wird die neue Sitzung vorbereitet. Dabei wird u.a. der Terminal-Typ (Environment-Variable $TERM) und die Art der Sitzung (interaktive Dialog-Sitzung oder Ausführung eines Kommandos) festgelegt.
- Zum Schluss startet der Server eine Login-Shell bzw. führt das gewünschte Kommando aus.

Interna der Verschlüsselung

Algorithmen bei asymmetrische Verschlüsselung

- RSA (Rivest, Shamir, Adleman)
  - Seit 20.10.2000 patentfrei
  - 1000mal langsamer als DES
  - Keylänge variabel
- DH (Diffie-Hellman)
- DSA (für digitale Signatur), von NSA entwickelt ;-)

Verwendung der Algorithmen durch SSH1/SSH2

- SSH1: RSA-Algorithmus und CRC-Check
- SSH2: Wegen Patentstreit um RSA entwickelt
  - DSA-Einsatz
  - HMAC-Hash-Funktion (Key hashing for Message Authentication)
- OpenSSH:
  - Unterstützt SSH1 und SSH2
  - Probleme bei Anmeldung SSH2 OpenSSH

Lizenzierung

- http://www.tectia.com
  Kommerzielle Lizenzen
- http://www.openssh.com
  Keine Einschränkungen, da BSD-Lizenz