

Konzepte von Betriebssystem-Komponenten:

SSH

Andre Lammel

andre.lammel@gmx.de

2002-06-17

1. Einführung

Oft sind auf einem entfernten Rechner administrative Aufgaben über unsichere Netze hinweg zu verrichten oder sensible Daten von dort zu einem lokalen Rechner oder umgekehrt zu übertragen. Hier können altbewährte Werkzeuge wie `rlogin`, `rcp`, `rsh`, `telnet` oder `ftp` den erforderlichen Schutz der übertragenen Daten durch Verschlüsselung und Integritätskontrolle nicht bieten. Die Daten werden als Plain Text, also für jedermann lesbar übertragen. Zum anderen sind die genannten Protokolle anfällig für Spoofing- und Man-In-The-Middle-Attacken.

1.1.SSH im Überblick

SSH ermöglicht sichere Kommunikation zwischen zwei Endpunkten durch unsichere Netzwerke über einen durch starke Kryptographie gesicherten Tunnel. Das Protokoll implementiert Schutz vor Abhören, Überwachung der Integrität, Serverauthentifizierung und Benutzerauthentifizierung. Zusätzliche Funktionen sind X11-Forwarding, TCP/IP-Forwarding und Komprimierung (zlib).

1.2.Historisches

1995 veröffentlichte Tatu Ylönen das erste Internet-Draft zum SSH Protokoll Version I. 1999 folgte Version 2 [2-5]. Es soll hier nur Version 2 des Protokolls vorgestellt werden, da Version 1 als unsicher gilt und in den aktuellen Implementierungen in der Hauptsache aus Kompatibilitätsgründen unterstützt wird.

2. Anatomie

In [5] ist die Architektur des Protokolls beschrieben. Grundsätzlich sind alle Algorithmen und die meisten Parameter verhandelbar. Jeder Algorithmus, der zum Einsatz kommt ist ersetzbar durch einen mit der selben Funktionalität; es können implementierungsspezifische Varianten oder verschiedene Versionen des gleichen Algorithmus verwendet werden. Um dies zu ermöglichen, wurde ein eigenes Schema für die Bezeichnungen der Algorithmen eingeführt und die völlige Unabhängigkeit der einzelnen Algorithmen innerhalb der jeweiligen Implementierung gefordert [3,5].

2.1.Struktur

Das SSH Protokoll ist in 4 Internet-Drafts [2-5] von Tatu Ylönen beschrieben und läßt sich in 3 Schichten mit jeweils eigenen Aufgaben untergliedern. Die Kommunikation ist paketorientiert und kann auf der Basis eines jeden Protokolls erfolgen, das die Übertragung von Binärdaten unterstützt und Übertragungsfehler korrigieren kann.

Eine Besonderheit ist, dass der Server sich gegenüber dem Client mittels eines Host Keys authentifizieren muß. Clientseitig kann der Key mittels zweier Verfahren überprüft werden:

- Dem Client liegt der Server Public Key in einer lokalen Datenbank (meist in der Datei `$HOME/.ssh/known_hosts`) schon vor und wird mit dem vom Server übertragenen Key verglichen.
- Der Client bedient sich einer externen Zertifizierung Authority (CA). In diesem Falle müsste der Server Public Key mit dem private Key der CA signiert sein und der Client würde mit dem Public Key der CA die Signatur des Server Keys überprüfen.

Kommt das erste Verfahren zum Einsatz, so besteht aus Gründen der leichteren Bedienbarkeit meist die Möglichkeit, den Key eines neuen, noch nicht besuchten Server automatisch in die lokale Datenbank einzutragen. Dies stellt zwar einen potentiellen Angriffspunkt dar, darauffolgende Sessions mit diesem Server wären aber wiederum sicher. Voraussetzung des zweiten Verfahrens ist die absolute Vertrauenswürdigkeit und Sicherheit der CA [16].

Mit der Direktive „StrictHostKeyChecking“ kann im Falle eines veränderten Server Keys eine Verbindung seitens des Clients automatisch unterbunden werden bzw. das Ergebnis der Überprüfung des Server Keys ignoriert werden.

2.2. Transport Layer

Der Transport Layer ist die erste aktive Instanz jeder SSH Verbindung, ein sicheres Transport Protokoll, dessen zentrales Format das Binary Packet Protocol ist. Es bietet starke Verschlüsselung, kryptographische Authentifizierung und gewährleistet die Integrität der übertragenen Daten. Übertragungsfehler werden als Angriff gewertet und führen zum Abbruch der Verbindung.

2.2.1. Das Binary Packet Protocol

Das Paketformat für das Binary Packet Protocol ist folgendermaßen definiert:

Typ	Name	Beschreibung
uint32	packet_length	Paketlänge ohne packet_lenght und mac
byte	padding_length	Länge der Fülldaten
byte[i]	payload	Nutzbarer Inhalt des Paketes (evtl. komprimiert)
byte[j]	padding	Fülldaten, so dass die Summe der Längen von packet_length, padding_length, payload und padding ein Vielfaches vom Maximum aus 8 und der cipher block size werden. Das Feld packet_length ist ebenfalls verschlüsselt.
byte[k]	mac	Message Authentication Code (MAC)

Die kleinste mögliche Paketlänge ist das Maximum aus 16 oder der cipher block size inclusive des MAC Bytes. Die konkreten Implementierungen sollten nach Empfang von Maximum aus 8 oder der cipher block length Bytes eines Paketes das Feld packet_length entschlüsseln.

Implementierungen sollten Pakete mit einer Payload der Länge 32768 Byte (unkomprimiert) verarbeiten können, wobei die Gesamtgröße des Paketes bis zu 35000 Byte betragen kann. Größere Pakete sollten nach Vereinbarung möglich sein.

2.2.2. Kommunikation

Nach dem Aufbau einer Verbindung über das zugrunde liegende Protokoll durch den Client senden beide Seiten einen Identifikationsstring in folgendem Format:

SSH-[Protokollversion]-[Softwareversion] [Kommentar]

Unterstützt der Server sowohl Version 1 als auch Version 2 der Protokolls, so muss dies dem Client durch die Angabe von 1.99 als serverseitige Protokollversion mitgeteilt werden. Unterstützt der Client Version 2, so muß er diese Angabe als Version 2 interpretieren. Für Clients, die nur Version 1 unterstützen, schaltet der Server in den Kompatibilitätsmodus. Sämtliche weitere Kommunikation erfolgt innerhalb des Binary Packet Protokolls.

2.2.3. Algorithmen und Schlüsseltausch

Zunächst einigen sich beide Partner mithilfe eines SSH_MSG_KEXINIT Paketes [3] auf eine Kombination von Algorithmen für den Schlüsseltausch, die Verschlüsselung und die Berechnung des MAC. Das Paket enthält eine Liste der jeweils unterstützten Algorithmen, von denen der erste bevorzugt zum Einsatz kommen sollte. Da dies von beiden Seiten erwartet wird, kann sofort im Anschluß jeder Partner das Paket zum Schlüsseltausch senden. Es muß verworfen werden, wenn sich die real benutzten Algorithmen von den geschätzten Algorithmen unterscheiden. Ein neues Paket unter der Verwendung der richtigen Algorithmen muß gesendet werden. In der Regel erfordert der Schlüsseltausch also nur zwei Pakete, maximal drei Pakete.

Die Algorithmen für den Hinweg und den Rückweg können unabhängig voneinander gewählt werden, ebenso die Kompression (zlib oder none). Für die Verschlüsselung und für den MAC kann als Algorithmus „none“ vereinbart werden, was jedoch zur Folge hat daß entweder alle Daten unverschlüsselt übertragen werden oder die Integrität der Daten nicht überprüft werden kann; auch beides ist möglich. In allen drei Fällen ist die Verwendung von SSH sinnlos, da die Probleme der oben genannten Werkzeuge, zumindest teilweise, auch hier bestehen würden.

Algorithmen für die Kommunikation zwischen Client und Server [1,10]	
Kompression	none, zlib
Cipher für die Verschlüsselung	3des-cbc, blowfish-cbc, twofish256/192/128-cbc, twofish-cbc, aes256/192/128-cbc, serpent256/192/128-cbc, arcfour, idea-cbc, cast128-cbc, none
Integrität (MAC)	hmac-sha1, hmac-sha1-96, hmac-md5, hmac-md5-96, none
Schlüsseltausch	diffie-hellman-group-sha1
Public Key Algorithmen	ssh-dss, ssh-rsa, x509v3-sign-rsa, x509v3-sign-dss, spki-sign-rsa, spki-sign-dss, pgp-sign-rsa, pgp-sign-dss

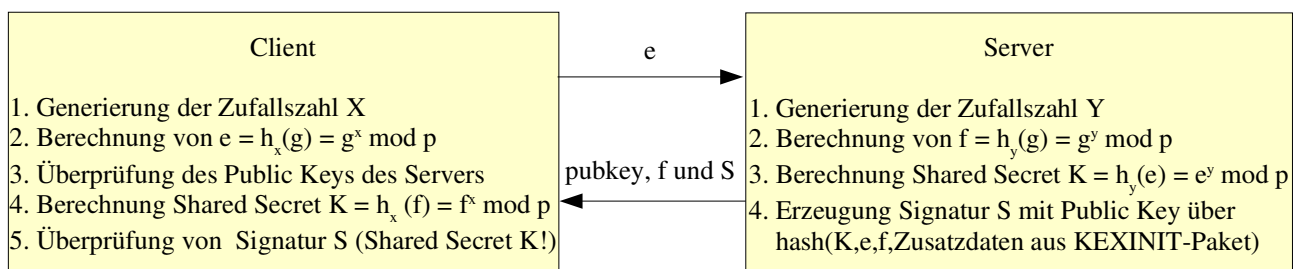
Ziel des Schlüsseltausches ist es, daß beide Seiten einen geheimen Schlüssel besitzen der nur den Kommunikationspartnern bekannt ist und für Dritte nur mit sehr großem Aufwand berechenbar ist. SSH verwendet hierfür den Diffie-Hellman-Algorithmus mit SHA1 als Hashfunktion (unabhängig von der für die Datenübertragung benutzten Hashfunktion!). Grundlage ist die Funktion:

$$h = h_x(g) = g^x \text{ mod } p \quad \text{mit } g=1, x = \text{Zufallszahl und } p = 2^{1024} - 2^{960} - 1 + 2^{64} \text{ floor}(2^{894} \quad 129093)$$

Bei Kenntnis von $h(g)$, g und p kann nur mit sehr großem Aufwand auf x geschlossen werden. Für zwei Primzahlen x und y ist die Anwendung von h_x und h_y kommutativ (vereinfacht, genauer unter [17]):

$$h_y(h_x(g)) = h_x(h_y(g))$$

Der eigentliche Schlüsseltausch wird nach folgendem Schema durchgeführt:



Das Ergebnis des Schlüsseltausches sind ein Shared Secret K und ein Exchange Hash H (Signatur S), der zugleich Sessionid ist. Daraus werden im Folgenden die Schlüssel für die Verschlüsselung in beiden Richtungen berechnet, wobei HASH (sha1) die Hash-Funktion aus dem Schlüsseltausch ist („+“ meint hier Konkatenation):

Schlüssel $K_n = \text{HASH}(K + H + B + \text{sessionid})$ mit $B = \text{bekannter Wert, z.B. „A“}$.

Die Daten für den jeweiligen Schlüssel werden ab dem ersten Bit des Ergebnisses von HASH() entnommen, für Algorithmen mit variabler Schlüssellänge sollten zumindest 128 bit verwendet werden. Wenn die Länge des Schlüsselmaterials K_n nicht ausreicht, wird das schon berechnete Schlüsselmaterial K_n durch das Anhängen des Ergebnisses einer weiteren Ausführung von HASH() erweitert:

$$\begin{aligned}
 K_1 &= \text{HASH}(K_n + H + B + \text{sessionid}) \\
 K_2 &= \text{HASH}(K_n + H + K_1) \\
 K_3 &= \text{HASH}(K_n + H + K_1 + K_2) \dots
 \end{aligned}$$

Dieser Vorgang wird so oft wiederholt, bis das zur Verfügung stehende Material für den Schlüssel ausreicht. Der Schlüsselaustausch endet mit einem SSH_MSG_NEWKEYS Paket [3].

Die Verschlüsselung der Pakete erfolgt auf der Ebene des Binary Packet Protocols mit den beiderseits ausgehandelten Verschlüsselungsalgorithmen und den vorher aus dem Shared Secret berechneten Schlüsseln. Der MAC wird mithilfe der vereinbarten MAC Algorithmen vor der Verschlüsselung des Paketes unter Anwendung auf das unverschlüsselte Paket, den aus dem Shared Secret gewonnenen Key und die laufende Nummer (sequence number) des jeweiligen

Paketes berechnet. Als einziges wird der MAC unverschlüsselt im letzten Feld des Paketes übertragen. Der Empfänger kann, indem er den MAC mit dem von der Gegenseite verwendeten Algorithmus über die entschlüsselten Daten erneut berechnet, die Integrität der empfangenen Daten überprüfen. Wurde Kompression vereinbart, so wird nur die Payload komprimiert. Der MAC wird dann unter Verwendung der komprimierten Daten berechnet, ebenso erfolgt die Verschlüsselung nach der Kompression. Ein erneuter Schlüsseltausch kann jederzeit von beiden Seiten eingeleitet werden. Der Status der darüberliegenden Schichten bleibt hierbei unverändert, die Sessionid bleibt erhalten.

Als nächstes schickt der Client ein `SSH_MSG_SERVICE_REQUEST` Paket [3], in dem einer der folgenden Services angefordert wird: `ssh-userauth`, `ssh-connection`. In den meisten Fällen ist der angeforderte Service `ssh-userauth`, um den Client zu authentifizieren.

2.2.4.Sicherheit

Die größte Gefahr geht neben der bekannten Unsicherheit beim Diffie-Hellman-Algorithmus von den im cipher block chaining (CBC) Modus benutzten Algorithmen aus. Es kann hier auf den Originaltext geschlossen werden, wenn zwei identische Datenblöcke mit dem selben Schlüssel verschlüsselt wurden. Aus diesem Grunde wird auch die Schlüsselerneuerung nach etwa einem Gigabyte übertragener Daten oder nach Ablauf einer Stunde empfohlen [3].

2.3.Authentication Layer

Der Authentication Layer ist ein allgemeines Protokoll, um Benutzer zu authentifizieren und für den Einsatz über dem SSH Transport Layer gedacht. Es verlässt sich darauf, dass das darunter liegende Protokoll Datenintegrität und Verschlüsselung implementiert.

2.3.1.Kommunikation

Der Authentication Layer startet unmittelbar nachdem der Client ein `SSH_MSG_SERVICE_REQUEST` Paket [4] mit der Anforderung „`ssh-userauth`“ an den Server geschickt hat. Vom darunter liegenden Protokoll werden die Sessionid in Form des Exchange Hash aus dem ersten Schlüsseltausch und Angaben, ob Verschlüsselung und Integritätskontrolle aktiv sind, erwartet.

Jede Authentifizierung startet mit einem `SSH_MSG_USERAUTH_REQUEST` Paket [4]. Der Server teilt dem Client zuerst mit, welche Methoden der Authentifizierung er unterstützt, woraufhin der Client die freie Wahl hat, sich für eine davon zu entscheiden. Zusätzlich teilt der Client den Namen des nach der Authentifizierung gewünschten Services mit. Wurde eine Methode ausgehandelt, so werden die erforderlichen Daten ausgetauscht. Mögliche Antworten sind `SSH_MSG_USERAUTH_SUCCESS` (Authentifizierung akzeptiert), `SSH_MSG_USERAUTH_FAILURE` Paket mit `partial_success=false` (Authentifikation nicht akzeptiert, Ende des Dialoges) und `SSH_MSG_USERAUTH_FAILURE` mit `partial_success=true` (Authentifikation akzeptiert, aber nicht ausreichend. Server sendet Liste der noch benötigten Verfahren) [4]. Ist die Authentifizierung komplett, startet der Server den angeforderten Service. Schlägt die Authentifizierung fehl, so muss der Server die Verbindung beenden.

Die Authentifizierung über Public Key ist die einzige, die als „`REQUIRED`“ [4], also unbedingt erforderlich betrachtet wird. Die Authentifizierung über Passwort oder Hostkeys auf dem entfernten Rechner sowie der Request für einen Passwortwechsel sind ebenfalls möglich.

2.3.2.Public Key Authentifizierung

Wird die Authentifizierung über einen Public Key gewählt, hat der Client, um unnötigen Rechenaufwand zu vermeiden, die Möglichkeit, vorher mit einem `SSH_MSG_USERAUTH_REQUEST` Paket, das nur den Public Key des Benutzers enthält, beim Server anzufragen, ob diese Methode in der gewünschten Schlüssel-Algorithmus-Kombination unterstützt wird. Ist der Key geeignet, so erzeugt der Client ein zweites Paket mit dem Public Key und einer Signatur, die mit dem Private Key des Benutzers erzeugt wurde, evtl. unter Abfrage des entspr. Passwortes des Benutzers. Der Server überprüft den Key und die Signatur; nur wenn beide korrekt sind, ist diese Methode erfolgreich.

2.3.3.Passwortauthentifizierung

Der Client schickt einen Request mit dem Benutzernamen und dem Passwort als Plain Text auf dem entfernten Rechner (durch das darunterliegende Protokoll wird bereits verschlüsselt!). Alternativ kann der Client einen Passwortwechsel anfordern, bzw. der Server kann dies tun, wenn das Passwort des Benutzers auf dem entfernten Rechner abgelaufen ist. Fordert der Server einen Passwortwechsel an, so kann der Client alternativ dazu auch eine andere Authentifizierungsmethode anfordern.

2.3.4.Hostbasierte Authentifizierung

Bei der hostbasierten Authentifikation wird anstatt des Private Keys des Benutzers der der Private Key des Clientrechners verwendet. Diese Methode ist ähnlich den herkömmlichen Konfigurationen mit `.rhosts` oder `hosts.equiv`.

2.3.6.Sicherheit

Die größte Unsicherheit geht von der hostbasierten Authentifikation aus, da hier unter Umständen der Benutzer Zugriff auf den privaten Schlüssel des Clientrechners erlangen kann und serverseitig keine Informationen zum Benutzer vorliegen. Der private Schlüssel des Rechners sollte mit den Rechten „400“ versehen werden und den Benutzer Root zum Eigentümer haben. Bei der Passwortauthentifizierung besteht die Gefahr unsicherer Passwörter und der Kompromittierung des darunterliegenden Protokolls durch Fehlkonfiguration seitens des Administrators oder einen Angriff eines lokalen Benutzers auf die Serverkonfiguration.

2.4.Connection Layer

Der Connection Layer ist die oberste Schicht über Transport- und Authentication Layer. Er implementiert interaktive Loginsessions, Ausführung von Befehlen auf entfernten Rechnern, TCP/IP Forwarding und X11 Forwarding. Hier werden die einzelnen Verbindungen zwischen den Kommunikationspartnern in Form von Channels gesteuert. Der Servicename ist „ssh-connection“.

2.4.1.Kommunikation

Die Kommunikation der Verbindungspartner findet innerhalb einzelner Channels mit jeweils einer speziellen Funktionen statt. Alle Channels werden über eine Verbindung gemultiplext und über Requests geöffnet, gesteuert oder geschlossen.

2.4.2.Channels

Ein Channel wird über ein `SSH_MSG_CHANNEL_OPEN` Paket geöffnet, jeder Partner ordnet dem Channel eine sog. magic number zu, die den Channel eindeutig indentifiziert, solange dieser offen ist. Mit Hilfe dieser Nummer erfolgt die Zuordnung der Requests zu den Channels. Die Datenübertragung erfolgt unter aktiver Flußkontrolle, wozu der folgende Algorithmus verwendet wird:

Beim Öffnen eines Channels wird eine Windowsize festgelegt. Jeder der Partner kann maximal Windowsize Bytes an Daten verschicken und muss dann auf ein `SSH_MSG_CHANNEL_WINDOW_ADJUST` vom Gegenüber warten. Erst dann können weitere `SSH_MSG_CHANNEL_WINDOW_ADJUST + [initialer Wert] = n` Bytes an Daten verschickt werden.

Es sind folgende Channels in der Protokolldefinition vorgesehen:

- Interaktive Login Session mit Shell (session)
- X11 Forwarding [14] (x11)
- Befehlsausführung, keine Shell (exec)
- Aufruf eines Subsystems wie sftp (subsystem)
- TCP/IP Forwarding (entfernter Serverport auf lokalen Port) (forwarded-tcpip)
- Lokales TCP/IP Forwarding (lokaler Port auf entfernten Server port) (direct-tcpip)

Die Eigenschaften der einzelnen Channels werden über channelspezifische Requests kontrolliert, deren detaillierte Erklärung [2] den Rahmen dieses Vortrages sprengen würde.

2.4.3.Sicherheit

Die drei schwerwiegendsten Sicherheitsrisiken liegen in der Remote Befehlsausführung, dem Setzen von Umgebungsvariablen und im TCP/IP Forwarding. Da der SSH Server als Benutzer Root gestartet werden muss, ist es hier erforderlich, serverseitig zu überprüfen, ob der entfernte Benutzer berechtigt ist den entsprechenden Befehl auf dem lokalen System auszuführen, und die passenden Umgebungsvariablen zu setzen. Das Internet-Draft sieht die Möglichkeit zur Befehlsausführung in beide Richtungen vor, empfiehlt jedoch, diese clientseitig zu unterbinden. Wird dies nicht getan, so besteht die Möglichkeit, den Client vom Server aus anzugreifen. Ebenso kann das X11-Forwarding einem Angreifer ermöglichen, Zugang über eine Firewall hinweg zu einem entfernten X-Server zu erhalten. TCP/IP Forwarding kann ebenfalls ausgenutzt werden, um über eine Firewall hinweg Zugang zu einem

geschützten Netzwerk zu erhalten. Seitens der Firewall ist es nicht möglich, den Inhalt der übertragenen Daten zu überwachen, da es sich um einen verschlüsselten Tunnel handelt. Es ist die Aufgabe des Administrators, solche Gefahren durch eine sinnvolle Konfiguration von Client und Server abzuwenden. Mehr dazu: [0]

3. Administration

3.1. Installation und Konfiguration

SSH ist vom OpenBSD Projekt im Quellcode erhältlich und Bestandteil aller Linux/BSD Distributionen. Bei der Konfiguration des Servers sollte darauf geachtet werden, dass hostbasierte Authentifizierung nicht möglich ist, ebenso wie die Authentifizierung allein anhand des Benutzernamens. Die sicherste Variante ist die Authentifizierung mit Hilfe von passwortgeschützten DSA-Keys. Hier wird sowohl das Passwort für den Schlüssel, als auch der Schlüssel selbst benötigt. Ebenfalls unterbunden werden sollten X11 Forwarding und TCP/IP Forwarding. Die obengenannten Funktionen sollten, wenn sie benötigt werden, nur dann zum Einsatz kommen, wenn sich sowohl Server als auch Client im selben, sicheren Subnetz befinden und keine Möglichkeit des Angriffes durch Dritte besteht.

3.2. Keys

Momentan werden RSA- und DSA-Keys in verschiedenen Formaten unterstützt:

- ssh (einfaches, SSH eigenes Standardformat, OpenSSH)
- x509v3 (X.509 Zertifikate Version 3)[11]
- spki (SPKI Zertifikate)
- pgp-sign (OpenPGP Zertifikate)[12]

Am häufigsten kommt das Standardformat zum Einsatz, daher wird auf die restlichen Formate hier nicht näher eingegangen. Das Standardformat kann mithilfe des zur SSH-Protokollsuite gehörigen Tools „ssh-keygen“ erzeugt werden.

3.2.1. Sicherheit

Wie oben schon erwähnt, ist die Authentifizierung über Public Keys die sicherste Authentifizierungsmethode, da hier nicht nur ein sondern zwei Faktoren zusammenspielen und die Wahrscheinlichkeit eines erfolgreichen Angriffes halbiert wird. Zur Sicherheit von OpenSSH: [0]. Zur Sicherheit der verwendeten Algorithmen: [1,10]

3.3. SmartCards

Die aktuelle Implementierung von OpenSSH unterstützt auch die Authentifizierung auf der Basis eines Schlüssels, der auf einer SmartCard gespeichert wurde. Diese Möglichkeit ist für Unternehmen mit einem hohen Bedarf an Sicherheit von Interesse. Leider ist es mir nicht gelungen, die aktuelle Version mit SmartCard-Unterstützung zu kompilieren, da die erforderlichen Bibliotheken [13] sich noch in der Entwicklungsphase befinden. Die Schlüssel lassen sich jedoch theoretisch mit dem Keygenerator (ssh-keygen) sowohl auf der SmartCard abspeichern als auch wieder auslesen. Es kommen die selben Schlüssel und Formate zum Einsatz, die für die lokale Speicherung im Homeverzeichnis des Benutzers verwendet werden.

3.4. Anwendungsbeispiele:

Lokalen Port 25 (gandalf) auf Port 7000 eines Servers (hapatus) verfügbar machen:

```
$ > ssh -R 7000:gandalf:25 -N andre@hapatus
```

Port 25 eines Servers (hapatus) auf lokalem Port 7000 (gandalf) verfügbar machen:

```
$ > ssh -L 7000:hapatus:25 -N andre@hapatus
```

Eine X11-Applikation auf dem Server (hapatus) starten, Oberfläche lokal (gandalf):

```
$ > ssh andre@hapatus xterm
```

Lokale Datei /opt/texte/test.txt mit scp auf den Server (hapatus) nach /home/andre übertragen:

```
$ > scp /opt/texte/text.txt andre@hapatus:/home/andre
```

Entfernte Datei /opt/texte/text.txt (hapatus) mit scp nach lokalem Verzeichnis /home/andre übertragen:

```
$ > scp andre@hapatus:/opt/texte/text.txt /home/andre
```

4. Abschließende Bemerkungen

SSH bietet eine sichere Alternative mit vielen Zusatzfunktionen zu den altbewährten, aber unsicheren Werkzeugen des Administrators einer UNIX/Linux Umgebung. Leider hat aber auch z.B. die Implementierung OpenSSH eine lange Liste von Security Bugs aufzuweisen [0]. Es sollte daher immer darauf geachtet werden, daß eine aktuelle Version der Protokollsuite im Einsatz kommt. Ebenso muss bei der Konfiguration darauf geachtet werden, dass durch unüberlegte Einstellungen nicht die Sicherheit der beteiligten Rechner gefährdet wird. Abgesehen von den weiterhin bestehenden Problemen der einzelnen kryptographischen Algorithmen sowie des weiterhin notwendigen Schlüsseltausches, ist SSH2 ein universell einsetzbares Protokoll zur Kommunikation zwischen entfernten Rechnern über unsichere Verbindungen. SSH2 ist durch die Möglichkeit effizienter Implementierung der zum Einsatz kommenden Algorithmen und Dialoge für den alltäglichen Gebrauch ohne Performanceeinbußen nutzbar, lediglich beim Dateitransfer mit SCP oder SFTP fällt der Protokolloverhead und die zur Verschlüsselung erforderliche Rechenleistung ins Gewicht.

5. Quellenangabe

[0] OpenSSH <http://www.openssh.org>

[1] Schneier, B., "Applied Cryptography Second Edition: protocols algorithms and source in code in C", 1996

[2] <http://www.ietf.org/internet-drafts/draft-ietf-secsh-connect-15.txt>
SSH Connection Protocol, T. Rinne, T. Ylonen, Tero Kivinen, M Saarinen, Sami Lehtinen, 02/01/2002.

[3] <http://www.ietf.org/internet-drafts/draft-ietf-secsh-transport-14.txt>
SSH Transport Layer Protocol, T. Rinne, T. Ylonen, Tero Kivinen, Markku Juhani Saarinen, Sami Lehtinen, 03/25/2002.

[4] <http://www.ietf.org/internet-drafts/draft-ietf-secsh-userauth-15.txt>
SSH Authentication Protocol, T. Rinne, T. Ylonen, Tero Kivinen, M Saarinen, Sami Lehtinen, 03/04/2002

[5] <http://www.ietf.org/internet-drafts/draft-ietf-secsh-architecture-12.txt>
SSH Protocol Architecture, T. Rinne, T. Ylonen, Tero Kivinen, M Saarinen, Sami Lehtinen, 02/01/2002

[6] <http://www.openssh.org/manual.html>
Die aktuellen Manual Pages zu den Komponenten von OpenSSH

[7] Linux Unleashed, ???

[8] Linux Magazin 2002-05, Artikel S.56-61, Blick-dicht

[9] Linux Magazin 2002-07, Artikel S.70-75, Tunnel-Blick

[10] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. Handbook of Applied Cryptography.
CRC Press, 1996

[11] OpenSSL, <http://www.openssl.org/docs/>

[12] OpenPGP, <http://www.openpgp.org/>

[13] M.U.S.C.L.E., <http://www.linuxnet.com/software.html>

[14] The Xfree86 Project, Inc., <http://www.xfree.org/>

[15] OpenBSD Project, <http://www.openbsd.org/>

[16] Versign, Inc. <http://www.verisign.com/>

[17] Erklärung zum Diffie-Hellman-Algorithmus <http://www.compapp.dcu.ie/~ekeave.case3/Networks/Diffie.html>