

Secure Shell (ssh)

Seminar: Simulationen mit User Mode Linux

WS 2005/06

Thorsten Bormer¹

Universität Koblenz-Landau tbormer@uni-koblenz.de

1 Einführung

Unter SSH versteht man sowohl ein Protokoll, dessen Ziel es ist, eine sichere Kommunikation über eine unsichere Verbindung zu ermöglichen, als auch eine Menge von Programmen, die auf diesem Protokoll aufsetzen — u.a. `ssh` und `scp`. Diese Programme dienen zur Remote-Administration von Rechnern und ersetzen damit ihre älteren Varianten `telnet`, `rlogin`, `rcp`, etc.

Terminologie Die Unterscheidung der Protokollversion von der konkreten Implementierung erfolgt folgendermaßen: Protokolle tragen eine Bezeichnung wie SSH-1, Implementierungen werden durch die Notation SSH2 kenntlich gemacht.

Sofern nicht näher angegeben, wird von der Protokollversion SSH-2 ausgegangen. Als Beispielimplementierung dient, aufgrund der großen Verbreitung und der freien Verfügbarkeit durch die BSD-Lizenz, OpenSSH.

Aufbau der Ausarbeitung Die historische Entwicklung von SSH wird in Abschnitt 1.1 wiedergegeben. Abschnitt 2 erläutert den oben formulierten Sicherheitsbegriff für Kommunikationskanäle und gibt einen Überblick über die in SSH verwendeten kryptographischen Methoden. Die Struktur des SSH-Protokolls und der Ablauf einer `ssh`-Verbindung zeigt Abschnitt 3 — die Anwendung von SSH in der Praxis wird in Abschnitt 4 behandelt.

1.1 Entwicklung von SSH

Die erste Version des SSH-Protokolls (SSH-1 [22]) entstand an der Technischen Universität Helsinki durch Tatu Ylönen im Jahr 1995, wahrscheinlich als Reaktion darauf, dass im lokalen Universitätsnetzwerk Passwörter abgehört wurden. [5] Die ersten Versionen der Implementation dieses Protokolls durch Ylönen wurden als Freeware lizenziert — der Gründung der Firma SSH Communications Security durch Ylönen folgte die Veröffentlichung der Programme unter einer proprietären Lizenz, die letzte 'freie' Version trug die Versionsnummer 1.2.12. [5,7,1] Eine verbesserte Version des Protokolls erschien unter dem Namen SSH-2 im Jahr 1996 und brachte gegenüber der ersten Version u.a. einen modularen und flexibleren Aufbau, sowie bessere Absicherung gegen Manipulationen am Datenstrom durch Verwendung von message authentication codes

(MACs, siehe 2.3). Gleichzeitig wurden mit SSH-2 (RFC [20,18,21,19]) die als Standard verwendeten kryptographischen Algorithmen ersetzt, so dass nun nicht mehr der durch RSA-Security patentierte RSA-Algorithmus zur Anwendung kam, sondern der weltweit frei verfügbare DSA-Algorithmus, sowie der Diffie-Hellman-Schlüsselaustausch, dessen Patent 1997 auslief. [15]

Da die freie Verwendung von SSH ausgehend von Version 1.2.12 eingeschränkt war, begannen 1999 Entwickler des OpenBSD-Projekts auf der Basis von Björn Grönvalls OSSH, welches wiederum von der freien Originalversion Ylönens abstammte, an einer Weiterentwicklung der Implementierung. [7] Diese hielt im gleichen Jahr unter dem Namen OpenSSH Einzug in OpenBSD 2.6 und wird auch heute in vielen Systemen weiterhin genutzt — eine Liste findet sich auf der offiziellen Website von OpenSSH [1].

2 Kryptographischer Hintergrund

Sicherheitsbegriff

Die wichtigsten erwünschten Eigenschaften einer sicheren Kommunikation lassen sich unter folgenden Punkten zusammenfassen [15]:

- Geheimhaltung/Vertraulichkeit der Nachrichten (*secrecy/confidentiality*): Nur dazu Berechtigte sollen eine chiffrierte Nachricht lesen können. Eine weitere Einschränkung liefert die *perfect forward secrecy*: werden Nachrichten mit einem Schlüssel verschlüsselt, der für jede Kommunikation mit einem „Haupt“schlüssel vereinbart wurde, so reicht die alleinige Kenntnis des Haupt- oder aktuellen Schlüssels nicht aus, um früher chiffrierte Nachrichten zu dechiffrieren.
- Authentifizierung (*authentication*): die Herkunft einer Nachricht muss eindeutig zu bestimmen sein, dabei darf es auch nicht möglich sein, dass ein Dritter Nachrichten erzeugen kann, die eine gültige, aber nicht ihm gehörige Absenderkennung tragen.
- Datenintegrität (*integrity*): Nachträgliche Veränderungen an Nachrichten (d.h. nach Absenden) können erkannt werden.

Die folgenden Abschnitte zeigen Möglichkeiten auf, wie diese Eigenschaften mit Hilfe kryptographischer Methoden zu realisieren sind.

2.1 Verschlüsselung

Symmetrische Verschlüsselung Die Vertraulichkeit von Nachrichten kann durch Verschlüsselung gewährleistet werden. Die einfachste Variante ist die symmetrische Verschlüsselung — dabei bestimmt ein vorher zwischen den Kommunikationspartnern vereinbarter Schlüssel sowohl den Ver- als auch den Entschlüsselungsschritt. Beispiele symmetrischer Verschlüsselungsalgorithmen sind 3DES[12], blowfish[16] und AES[10].

Der Nachteil dieser Verfahren ist zum einen die Problematik der Übermittlung der Schlüssel, die nicht auf dem gleichen Kommunikationsmedium erfolgen kann, den die verschlüsselte Nachricht nutzt, da als Prämisse eine unsichere Verbindung vorausgesetzt wurde — zum anderen benötigt man für jedes Sender-Empfänger-Paar einen eigenen Schlüssel, so dass die Anzahl der Schlüssel quadratisch mit der Anzahl der Kommunikationspartner (n) steigt, da folgender Zusammenhang gilt:

$$|\text{schlüssel}| = \frac{n^2 - n}{2}$$

Für $n = 100$ beträgt die Anzahl benötigter Schlüssel bereits 4950. [9]

Diffie-Hellman Schlüsselaustausch Die 1976 von Martin Hellman und Whitfield Diffie veröffentlichte Methode [11] des Diffie-Hellman-Schlüsselaustauschs (im Folgenden: DH) ermöglicht es zwei Kommunikationspartnern, über eine unsichere Verbindung einen gemeinsamen, geheimen Schlüssel zu vereinbaren. Dabei können weder Dritte diesen Schlüssel in Erfahrung bringen, noch erfährt die Gegenseite den jeweils anderen, privaten Schlüssel, der zur Generierung des gemeinsamen Schlüssels notwendig ist.

Die beschriebene Methode nutzt die Eigenschaften so genannter *Einwegfunktionen* aus. Informell ist eine Funktion eine Einwegfunktion, wenn sich ein effizientes Verfahren angeben lässt, um den Funktionswert zu einem bestimmten Argument zu bestimmen, aber kein effizientes Verfahren bekannt ist, welches das Inverse zu einem Funktionswert liefert. Da kein Beweis für die Existenz solcher Einwegfunktionen existiert, muss bei der Auswahl von Funktionen auf empirische Daten zurückgegriffen werden. [6]

Die im DH verwendete Einwegfunktion ist die Exponentiation modulo p , wobei p eine Primzahl ist. Da p prim ist, bildet die Menge der Zahlen $\mathbb{M} = \{1, \dots, p-1\}$ zusammen mit der Multiplikation modulo p eine zyklische Gruppe, die multiplikative Gruppe modulo p (\mathbb{Z}_p^*). Sei g , ein erzeugendes Element von \mathbb{Z}_p^* , gegeben, dann lässt sich für ein Element $x \in \mathbb{M}$ die Exponentiation

$$f(x) = g^x \text{ mod } p$$

leicht berechnen — es ist aber kein effizienter Algorithmus bekannt, der zu gegebenem $f(x)$ das zugehörige x berechnet, den *diskreten Logarithmus* von $f(x)$ zur Basis g . [9]

Mit Hilfe dieser Funktion lässt sich der Schlüsselaustausch folgendermaßen realisieren (nach [9]):

1. Vorbedingung: Kommunikationspartner haben eine hinreichend große Primzahl p und ein erzeugendes Element g vereinbart — diese Information kann öffentlich zugänglich sein.
2. Sender und Empfänger wählen jeweils eine zufällige Zahl aus der Menge \mathbb{M} (im folgenden: x und y)
3. Sender und Empfänger berechnen $f(x) = g^x \text{ mod } p$ bzw. $f(y) = g^y \text{ mod } p$ und senden das Ergebnis an die Gegenseite

4. aus x und $f(y)$ bzw. y und $f(x)$ lässt sich

$$f(y)^x = (g^y)^x = g^{x \cdot y} = (g^x)^y = f(x)^y$$

berechnen. Der geheime Schlüssel ist damit $g^{x \cdot y}$.

2.2 Authentifizierung

Für die eindeutige Zuordnung einer Nachricht zu einem Sender und die Verifikation der Herkunft einer Nachricht reicht es nicht, diese zu verschlüsseln — dazu muss eine Authentifizierung erfolgen, z.B. durch *Signaturen*. Ein möglicher Ansatz zur Realisierung von Signaturfunktionen sind die Public-Key-Verfahren, die im Folgenden beschrieben werden.

Public-Key-Kryptographie Die Idee hinter der Public Key Kryptographie ist die Asymmetrie der Ver- und Entschlüsselung. Im Gegensatz zur symmetrischen Verschlüsselung, bei der ein geheimer Schlüssel sowohl den Verschlüsselungsschritt (mit E bezeichnet), als auch den Entschlüsselungsschritt (D) bestimmt, existieren bei Public-Key-Verfahren jeweils zwei Schlüssel für Sender und Empfänger: ein privater Schlüssel s , der als Parameter für den Entschlüsselungsprozess dient, sowie ein öffentlicher Schlüssel p , der zur Verschlüsselung benutzt wird. [6]

Eine sinnvolle Ver- und Entschlüsselungsfunktion muss nach [6] u.a. folgende Eigenschaften erfüllen:

- $D \circ E = id$, d.h. wird eine Nachricht M mit E verschlüsselt und danach der Dechiffrierschritt D auf das Resultat angewandt, so erhält man wieder die ursprüngliche Nachricht M
- der Aufwand, um aus dem öffentlichen Schlüssel den privaten Schlüssel zu berechnen ist so groß, dass es unwahrscheinlich ist, dass dies in einer festzulegenden Zeitspanne möglich ist

Signaturverfahren mit Public-Key-Kryptographie Gilt für die Ver- und Entschlüsselungsfunktion außer den oben genannten beiden Punkten noch die Eigenschaft, dass auch $E \circ D = id$ ist, so lässt sich damit eine Signatur einer Nachricht folgendermaßen erstellen:

1. der Sender wendet die „Dechiffrierfunktion“ D auf die zu authentifizierende Nachricht M an und erhält die Signatur s .
2. die Signatur zusammen mit der Nachricht M wird an den Empfänger gesendet
3. der Empfänger kann durch Anwendung von E auf s die Signatur überprüfen, da bei einer gültigen Signatur folgendes gelten muss:

$$E(s) = E(D(M)) = M \quad (\text{da } E \circ D = id)$$

Damit ist sichergestellt, dass der Sender der Nachricht im Besitz des privaten Schlüssels ist, der zu dem entsprechenden öffentlichen Schlüssel gehört — eine weitere Identifikation des Senders muss gesondert erfolgen, z.B. durch einmaligen Abgleich der öffentlichen Schlüssel über ein geeignetes Kommunikationsmedium.

2.3 Datenintegrität

Um Veränderungen an versendeten Nachrichten bemerken zu können, werden in SSH2 *Message Authentication Codes* (MAC) eingesetzt, die sich in der Laufzeit gegenüber der Integritätsprüfung mittels Public-Key-Signaturverfahren günstiger verhalten. [17]

Ein Message Authentication Code ist von der Funktion einer kryptographischen Hashfunktion ähnlich. Genau wie diese bildet ein MAC eine beliebig lange Zeichenkette auf eine Zeichenkette fester Länge ab, im Idealfall ist dies eine zufällige Verteilung von Eingabedaten zu Ausgabedaten. Die geforderte Eigenschaft solcher Funktionen ist die (starke) Kollisionsresistenz, d.h. es ist nur mit „großem“ Aufwand möglich, zwei Eingabedaten zu finden, für den die MAC/Hashfunktion das gleiche Resultat liefert. [17,9]

Zusätzlich zu den Nutzdaten im Falle der reinen Hashfunktion wird bei einem MAC ein geheimer Schlüssel K benötigt, der in die Berechnung mit einfließt. Durch diesen Zusatz ist es einem Angreifer nicht ohne weiteres möglich, zu einer Nachricht den MAC zu berechnen — verändert der Angreifer eine Nachricht, so wird dies durch die Inkonsistenz zwischen Nachricht und MAC erkannt.

3 Aufbau von SSH

Den grundlegenden Aufbau des SSH-Protokolls mit der Einteilung in die einzelnen Schichten zeigt Abb. 1 — dabei wird angenommen, dass das Protokoll, wie in [21] als Standard angegeben, auf TCP/IP läuft.

Die einzelnen Layer des SSH-Protokolls bieten auf der Grundlage von TCP/IP folgende Eigenschaften [20,21,19]:

- Transport Layer (SSH-TRANS): realisiert Geheimhaltung der Daten, sowie Datenintegrität und bietet optional Komprimierung der Daten an. Weiterhin wird in dieser Schicht der Server gegenüber dem Client authentifiziert.
- User Authentication Layer (SSH-USERAUTH): weist den Benutzer gegenüber dem Server aus.
- Connection Layer (SSH-CONNECT): ermöglicht mehrere logische Kanäle auf einer verschlüsselten Verbindung.

jeder Layer setzt dabei auf dem vorhergehenden auf.

3.1 Einbettung der kryptographischen Verfahren in SSH

Die oben genannten Eigenschaften der einzelnen Schichten von SSH werden durch die in Kap. 2 beschriebenen Verfahren realisiert — die konkret benutzten Methoden sind in den RFCs 4251[20], 4252[18] und 4253[21] beschrieben.

Dabei findet der DH-Schlüsselaustausch aus Kap. 2.1 Anwendung, um einen geheimen Schlüssel zu erzeugen. Dieser dient als Quelle für Schlüssel symmetrischer Verfahren — u.a. aus Effizienzgründen werden die Nutzdaten der SSH-Verbindung nicht mit asymmetrischen Verfahren verschlüsselt. [17]

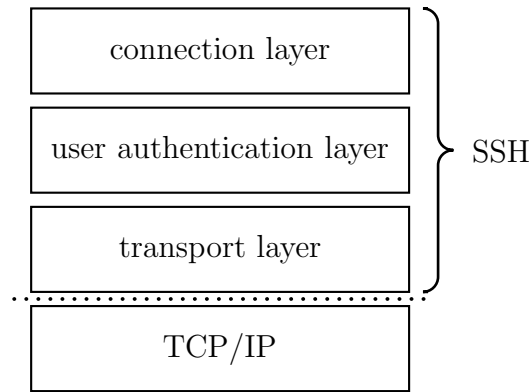


Abb. 1. Struktur des SSH-Protokolls

Die Authentifizierung des Servers gegenüber dem Client erfolgt durch ein Public-Key-Schlüsselpaar (host key) im Besitz des Servers. Ein Hashwert des öffentlichen Schlüssels ermöglicht dem Benutzer des Clients, die Zugehörigkeit des Schlüssels mit dem gewünschten Server zu überprüfen. Der Server kann dann per Signatur nachweisen, dass er in Besitz des zugehörigen privaten Schlüssels ist und identifiziert sich damit eindeutig.

Im umgekehrten Fall ist die Authentifizierung je nach Konfiguration des Servers durch mehrere Schritte möglich:

- Benutzername/Passwort: der Benutzer des Clients sendet dem Server Namen und Passwort eines Benutzers, der dem Server bekannt ist
- Public-Key (hostbasiert oder benutzerbasiert): der Client ist im Besitz eines Public-Key-Schlüsselpaars, welches dem Server durch den öffentlichen Schlüssel bekannt ist. Der Unterschied zwischen host- und benutzerbasierter Authentifikation ist die Granularität der Zuordnung und die Zugehörigkeit der Schlüssel (entweder zum jeweiligen Benutzer oder für den gesamten Host).

Durch die Verwendung von Message Authentication Codes wird schließlich, wie schon in Kap. 2.3 beschrieben, die Datenintegrität gewährleistet.

Die verwendeten Algorithmen für den Schlüsselaustausch, die Verschlüsselung, Datenintegrität, Kompression, sowie das angewandte Public-Key-Verfahren können zwischen dem Server und Client ausgehandelt werden, indem jede Seite eine Liste von verfügbaren Methoden im Klartext sendet. Bis auf den Schlüsselaustausch kann dabei in jede Richtung ein unterschiedlicher Algorithmus benutzt werden. Tabelle 1 zeigt eine Auswahl möglicher vordefinierter Optionen in den einzelnen Kategorien — weiterhin können über domainspezifische Namen des Formats `methodName@domain` zusätzliche Namen definiert werden, die lokal nutzbar sind.

Name	obligatorisch?	Kommentar
Kompression		
zlib	optional	LZ77-Kompression [23]
Verschlüsselung		
3des-cbc	ja	Triple-DES (CBC-Modus)
blowfish-cbc	empfohlen	Blowfish (CBC-Modus) [16]
twofish128-cbc	empfohlen	Twofish [14]
aes128-cbc	empfohlen	AES
Message Authentication Code		
hmac-sha1	ja	HMAC-SHA1 (Schlüssellänge 20)
Schlüsselaustausch		
diffie-hellman-group1-sha1	ja	DH-Schlüsselaustausch mit SHA-1 als Hash und defini- erter Gruppe
Public-Key-Verfahren		
ssh-dss	ja	DSS
ssh-rsa	empfohlen	RSA

Tabelle 1. Auswahl standardisierter Algorithmen in SSH, nach [21]

Ablauf einer SSH-Verbindung Die grundlegende Struktur einer SSH-Sitzung soll an einem vereinfachten Beispiel (auf der Grundlage der SSH-RFCs) dargestellt werden. Die chronologische Reihenfolge der Ereignisse, sowie deren Zugehörigkeit zu den einzelnen Schichten von SSH ist dabei:

- Transport Layer
1. Client stellt eine Verbindung zum Server her, im Normalfall über TCP/IP auf Port 22
 2. Client und Server tauschen Identifikationsstring der Form (`SSH-protocolVersion-softwareVersion-comments`) aus, im Fall von `linux.uni-koblenz.de` lautet dieser: `SSH-1.99-OpenSSH.2.9p1`
 3. Der DH-Schlüsselaustausch nach Kap. 2.1 wird durchgeführt; daraus resultiert der gemeinsame, geheime Schlüssel K . Weiterhin wird ein Hashwert H berechnet, der u.a. aus den Identifikationsstrings, dem öffentlichen Hostkey und einem zufällig gewählten Cookie gebildet wird.
 4. Aus K und H werden Schlüssel für die symmetrische Verschlüsselung bzw. für die Authentifizierung erstellt, H dient zudem als session identifier
 5. Server erstellt eine Signatur von H mit seinem privaten Host-Schlüssel und versendet das Ergebnis an den Client

- | | |
|---------------------------|--|
| | 6. Client schickt service request an Server, eine mögliche Anfrage ist hier z.B. „ssh-userauth“, um die Authentifizierung des Benutzers am Server einzuleiten |
| User Authentication Layer | <ol style="list-style-type: none"> 1. Server sendet eine Liste von Strings von verfügbaren Authentifizierungsmethoden 2. Client versucht gewünschte Authentifizierung zu erfüllen, indem er z.B. durch eine Signatur, erstellt mit einem private key, beweist, dass er im Besitz dieses privaten Schlüssels ist 3. Client wiederholt Schritt 2, bis Server genügend gültige Authentifizierungen erhalten hat, dies wird durch Pakete mit der Kennung SSH-MSG-USERAUTH-FAILURE und Liste der noch ausstehenden Schritte bzw. SSH-MSG-USERAUTH-SUCCESS signalisiert |
| Connection Layer | <ol style="list-style-type: none"> 1. Client oder Server öffnen einen Channel (SSH_MSG_CHANNEL_OPEN) 2. nach positiver Bestätigung der Gegenseite können Nutzdaten gesendet werden (SSH_MSG_CHANNEL_DATA) 3. Ende der Datenübertragung und Ende der Verbindung wird mit SSH_MSG_CHANNEL_EOF- und SSH_MSG_CHANNEL_CLOSE-Paketen signalisiert |

4 SSH in der Praxis

Dieser Abschnitt gibt eine Übersicht über die Funktionalität und Benutzung einer Auswahl von Programmen des ssh-Pakets. Die gezeigte Syntax der Kommandos wurde der OpenSSH-Website[1] und den manpages der Programme entnommen — dort findet sich auch eine ausführliche Beschreibung aller im Weiteren nicht vorgestellten Clients.

4.1 Benutzung des ssh-Clients

Syntax der Befehle Der ssh-Client dient dazu, auf entfernten Rechnern über ein Netzwerk einen Kommandozeileninterpreter bereitzustellen oder Befehle auf diesem Rechner auszuführen und soll damit u.a. Telnet, rlogin und rsh ersetzen.

Ein einfaches Beispiel der Syntax von ssh, bei der sich ein lokaler Benutzer auf dem Server `linux.uni-koblenz.de` unter gleichem Benutzernamen anmelden will, ist:

```
ssh linux.uni-koblenz.de
```

Entspricht der lokale Benutzername nicht demjenigen des entfernten Systems, so muss der Name explizit angegeben werden, indem dieser mit einem „@“ abgetrennt vor den Namen des Servers geschrieben wird (z.B.: `bob:/$ ssh alice@linux.uni-koblenz.de`)

Einzelne Befehle können mit ssh auf dem entfernten Rechner ausgeführt werden, indem diese am Ende der Kommandozeile des ssh-Aufrufs als Parameter angefügt werden — soll z. B. das Programm `/bin/doSomeCommand` auf `linux.uni-koblenz.de` zur Ausführung kommen, so lautet der Befehl:

```
ssh linux.uni-koblenz.de /bin/doSomeCommand
```

Authentifizierung des Servers Bei der erstmaligen Verbindung zu einem Server soll der Client nach [20] einen Fingerprint des öffentlichen Schlüssels des Hostkeys anzeigen, der nach Bestätigung lokal gespeichert wird und bei weiteren Verbindungen als Vergleich dient — im Falle von `linux.uni-koblenz.de` ergibt sich folgende Ausgabe:

```
ssh linux.uni-koblenz.de

The authenticity of host 'linux.uni-koblenz.de
(141.26.64.104)' can't be established.
RSA key fingerprint is
63:20:bd:5d:c8:e2:c7:12:93:76:09:d3:27:3e:d2:a2.
Are you sure you want to continue connecting
(yes/no)? yes
Warning: Permanently added 'linux.uni-koblenz.de,
141.26.64.104' (RSA) to the list of known hosts.
```

Unterbleibt die Überprüfung des Fingerprints an einer glaubwürdigen Quelle des öffentlichen Schlüssels vom Server, so besteht die Gefahr eines Man-in-the-Middle-Angriffs, dargestellt in Beispiel 1.

Beispiel 1. Client A will eine Verbindung mit Server B herstellen. Dem Angreifer C sei es möglich, beliebige Daten der Verbindung abzufangen und zu ändern. Damit kann sich C als Server B ausgeben, indem er den Fingerprint eines frei gewählten Public-Key-Schlüsselpaars an A sendet. Weiterhin stellt C eine normale SSH-Verbindung zum Server her — dadurch kann der Angreifer sowohl jede Nachricht in beide Richtungen mitlesen, als auch aktiv verändern, da ihm alle notwendigen Schlüssel bekannt sind. Abbildung 2 verdeutlicht diese Situation.

Benutzerauthentifizierung mit Public Keys Wie in Abschnitt 3.1 erwähnt, besteht neben der Möglichkeit den Benutzer des Clients durch Angabe von Benutzername und Passwort am Server zu identifizieren, die Variante der Authentifikation mittels Public Key. Dabei besitzt jeder Benutzer des Clients ein Public-Key-Schlüsselpaar.

Die dafür benötigten Schlüssel können mit Hilfe von `ssh-keygen` generiert werden. Die Länge des erzeugten Schlüsselpaars und die Art des Public Keys lässt sich durch die Optionen `-b` und der Angabe der Länge in Bits, sowie `-t` gefolgt von einem Kürzel für den Typ des Keys (z.B. „dsa“ oder „rsa“) steuern. Zur Generierung eines DSA-Schlüsselpaars mit 2048 Bit Länge lautet der Befehl also:

```
ssh-keygen -b 2048 -t dsa
```

Nach Generierung des Schlüsselpaars ist der Anwender angehalten, ein Passwort zu vergeben, mit dem der geheime Schlüssel verschlüsselt wird, um eventuelles Auslesen des lokal gespeicherten Keys von Unbefugten zu verhindern. Bei Verwendung des geheimen Schlüssels durch den ssh-Client wird das

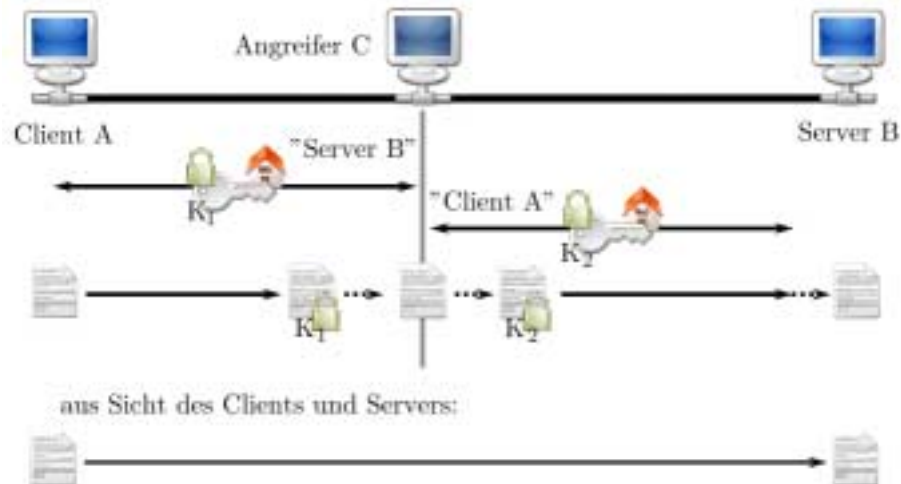


Abb. 2. Man-in-the-Middle-Angriff. Angreifer C ist an der Generierung beider Schlüssel K_1 und K_2 beteiligt und kann daher beliebig Nachrichten ver- und entschlüsseln.

Passwort dementsprechend benötigt, um den geheimen Schlüssel zu dechiffrieren.

Um die Authentifizierungsmethode per PK nutzen zu können, muss der öffentliche Teil eines so generierten Schlüssels dem Server mitgeteilt werden, indem der Benutzer diesen an die Datei `~/.ssh/authorized_keys` in seinem home-Verzeichnis auf dem Server anhängt. Unterstützt dieser die Authentifizierungsmethode mit Public Keys, so wird beim Login durch den Client der Nachweis erbracht, dass der Benutzer in Besitz des entsprechenden geheimen Schlüssels ist.

Port/X11-Forwarding Eine weitere Funktion von ssh ist das *Port Forwarding* — damit ist es möglich, eine TCP/IP-Verbindung über einen, von ssh bereitgestellten, verschlüsselten Kanal zu übertragen (zu „tunneln“).

Ein lokales Forwarding erstellt man mit der Anweisung

```
ssh -L lokalerPort:S:remotePort benutzer@hostname
```

Dadurch verbindet sich der lokale ssh-Client mit `hostname` (unter dem Benutzernamen `benutzer`) und öffnet einen Kanal, der alle lokal auf Port `lokalerPort` ankommenden TCP-Verbindungen auf den ssh-Server auf `hostname` weiterleitet — dieser wird angewiesen die Pakete zum Server `S` auf den Port `remotePort` zu senden. Beispiel 2 zeigt einen Anwendungsfall für lokales Portforwarding.

Beispiel 2. Eine gesicherte Verbindung zwischen einem lokalen Host `H` soll mit `news.uni-koblenz.de` hergestellt werden, damit ein Newsclient (auf `H`) den

Newsserver nutzen kann, ohne sensible Informationen im Klartext versenden zu müssen. Beide Rechner sind über TCP/IP miteinander verbunden. Der Befehl für das Erstellen des lokalen Forwardings lautet

```
ssh -L 1025:news.uni-koblenz.de:119 user@news.uni-koblenz.de
```

und muss auf H ausgeführt werden. Danach muss der Newsclient angewiesen werden, als Server localhost mit Port 1025 zu benutzen — die Weiterleitung der Pakete erfolgt dann gemäss Abbildung 3.



Abb. 3. Vereinfachter Ablauf einer lokalen Weiterleitung von Port 1025 (localhost) nach news.uni-koblenz.de, Port 119

In gleicher Weise kann auch die Netzwerkverbindung des unter Unix gebräuchlichen X Window Systems durch SSH getunnelt werden. Das X-System besteht aus einem X-Server, der die Ansteuerung der Ein- und Ausgabegeräte übernimmt und beliebig vielen Anwendungsprogrammen als X-Clients. Diese Programme kommunizieren über ein spezielles Protokoll, so dass es möglich ist, Anwendungsprogramm und X-Server auf unterschiedlichen Rechnern zu betreiben — allerdings erfolgt diese Kommunikation unverschlüsselt. Mit Hilfe des X11-Forwardings von ssh kann dies über eine gesicherte Verbindung geschehen. Der Aufruf von ssh zum Login am Server erfolgt hier mit der Option `-X`, um X11-Forwarding explizit zu erlauben, also z.B.:

```
ssh -X linux.uni-koblenz.de
```

Wird im Kommandozeileninterpreter ein X-Client gestartet, z.B. `xterm`, so erscheint die grafische Ausgabe auf dem X-Server des lokalen Hosts, während die Anwendung auf dem Server läuft.

Konfiguration des ssh-Servers Das Verhalten des ssh-Servers kann sowohl über Kommandozeilenparameter als auch über eine Konfigurationsdatei (im Normalfall `/etc/sshd_config` für OpenSSH) beeinflusst werden. Abbildung 4 zeigt eine Übersicht einiger wichtiger Optionen aus der Konfigurationsdatei.

Über die Option `Port x` lässt sich der Port einstellen, auf den der ssh-Server Verbindungen akzeptiert. Die Parameter `AllowUsers` und `DenyUsers`, sowie die

Pendants `DenyGroups` und `AllowGroups` mit Angabe einer Liste von Benutzern/-Gruppen ermöglichen es, bestimmten Benutzern und Gruppen explizit die Anmeldung am Server zu gewähren (und damit alle anderen nicht zuzulassen) bzw. zu verweigern — die Namen der Benutzer/Gruppen können dabei auch die Platzhalter `*` (für beliebige Zeichenketten) und `?` (beliebiges Zeichen) enthalten. Ein Sonderfall der Einschränkung von Berechtigungen ist `PermitRootLogin`, welches angibt, ob es dem Benutzer `root` erlaubt ist, über `ssh` auf den Server zuzugreifen. Weiterhin kann die Art der Authentifizierung durch `HostbasedAuthentication` und `PasswordAuthentication` bestimmt werden, um zusammen mit der Option `PermitEmptyPasswords` dem gewünschten Sicherheitsanspruch an die `ssh`-Verbindung gerecht zu werden.

```
Port 22 [22]
AllowUsers "bob" [all users]
DenyUsers "eve" [none]
DenyGroups [none]
AllowGroups [all groups]
PermitRootLogin "no" [yes]
HostbasedAuthentication "no" [no]
PasswordAuthentication "yes" [yes]
PermitEmptyPasswords "no" [no]
AllowTcpForwarding "yes" [yes]
X11Forwarding "yes" [no]
```

Abb. 4. Beispielinhalt der Konfigurationsdatei des `ssh`-Servers. Werte in eckiger Klammer sind nicht Bestandteil der Datei, sondern zeigen die Standard-Voreinstellung des Servers an.

4.2 scp

Eine weitere Applikation, die auf dem `SSH`-Protokoll aufbaut ist `scp`. Mit diesem Programm können über einen sicheren Kanal Dateien zwischen zwei Rechnern übertragen werden. Der Ablauf des Programms erfordert dabei keine Interaktion des Benutzers bis auf die eventuell nötigen Abfragen von Passwörtern zur Benutzerauthentifikation, zeigt aber den aktuellen Stand der Übertragung als Balkengraph an.

Die grundlegende Syntax des Programms ist: `scp Quelle Ziel`. Dabei kann sowohl bei „Quelle“ als auch bei „Ziel“ der Host explizit angegeben werden, mit der Syntax: `[[benutzer@]host:]datei` — damit ist auch ein Dateitransfer zwischen zwei entfernten Rechnern möglich. Möchte also Benutzer `bob` die Datei `datei.txt` in seinem home-Verzeichnis auf dem Server `linux.uni-koblenz.de` in das aktuelle, lokale Verzeichnis kopieren, so entspricht dies dem Kommando:

```
scp bob@linux.uni-koblenz.de:datei.txt ./
```

Neben den hier exemplarisch vorgestellten Programmen und den restlichen, zum ssh-„Paket“ zugerechneten Anwendungen wie z.B. `sftp`, existieren eine Vielzahl an Applikationen, die den SSH-Dienst nutzen oder dessen Nutzung empfehlen, dabei sind z.B. VNC oder `x2x`[3] zu nennen.

5 Fazit

Die stetig steigende Zahl der Internetnutzer, sowie der Einsatz neuer Dienste und Verbindungsmöglichkeiten führen zu neuen Problemen. Ersteres erhöht die Wahrscheinlichkeit anfällige Systeme vorzufinden, letzteres birgt das Risiko neuer Sicherheitslücken. Zu nennen sind hier z.B. Funknetzwerke, die Anfangs den Verschlüsselungsstandard *Wired Equivalent Privacy* (WEP) benutzen, festgelegt als Teil des IEEE 802.11-Standards[4]. Da die Unsicherheit der Verschlüsselungsmethode nachgewiesen werden konnte [8], gilt diese als überholt und sollte nicht mehr zum Einsatz kommen.

Dies zeigt, dass der Einsatz von SSH zum Zweck sicherer Kommunikation keinesfalls überflüssig geworden ist — ein Indiz dafür mag auch die steigende Anzahl der SSH-Server sein im Vergleich zur Anzahl der erreichbaren Hosts im Internet, eine Stichprobe dazu, mittels `ScanSSH`[2], liefert der Bericht [13].

Literatur

1. Openssh project website. <http://www.openssh.org/>.
2. Scanssh project website. <http://www.monkey.org/provos/scanssh/>.
3. x2x project website. <http://x2x.dottedmag.net/>.
4. *IEEE Std 802.11, 1999 Edition Information technology- Telecommunications and information exchange between systems- Local and metropolitan area networks- Specific requirements- part 11: Wireless Lan Medium Access Control (MAC) And Physical Layer (PHY) Specifications*, 1999.
5. Daniel J. Barrett and Richard E. Silverman. *SSH: The Secure Shell: The Definitive Guide*. 2001.
6. Friedrich L. Bauer. *Kryptologie: Methoden und Maximen*. Springer-Verlag, 1993.
7. Michael D. Bauer. *Building Secure Servers with Linux*. 2002.
8. Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: The insecurity of 802.11. <http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html>, 2001.
9. Johannes Buchmann. *Einführung in die Kryptographie*. Springer-Verlag, 1999.
10. J. Daemen and V. Rijmen. Aes proposal: Rijndael.
11. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
12. National Institute of Standards and Technology. *FIPS PUB 46-3: Data Encryption Standard (DES)*. National Institute for Standards and Technology, Gaithersburg, MD, USA, October 1999. supersedes FIPS 46-2.
13. Niels Provos and Peter Honeyman. Scanssh - scanning the internet for ssh servers.
14. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. Two-fish: A 128-bit block cipher. Technical report, Counterpane Systems, 101 East Minnehaha Parkway, Minneapolis, MN 55419, June 1998.

15. Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1993.
16. Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption, Cambridge Security Workshop*, pages 191–204, London, UK, 1994. Springer-Verlag.
17. Niels Ferguson and Bruce Schneier. *Practical cryptography*. John Wiley & Sons, Inc., 2003.
18. T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol. RFC 4252 (Proposed Standard), January 2006.
19. T. Ylonen and C. Lonvick. The Secure Shell (SSH) Connection Protocol. RFC 4254 (Proposed Standard), January 2006.
20. T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), January 2006.
21. T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard), January 2006.
22. Tatu Ylonen. The ssh (secure shell) remote login protocol. internet-draft. draft-ylonen-ssh-protocol-00.txt (expired), 1996.
23. Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.