Remote-Ausführen von Kommandos — Analyse zur Verwendung von ssh —

Tino Schwarze, TeeGee

September 2000

Inhaltsverzeichnis

1 Anforderungen

- Authentifizierung (-> Keys etc.)
- Ausführung von Kommando(s)
- stdin/stdout über das Netz umleiten

⇒ Alle diese Anforderungen erfüllt `ssh`.

2 Wie funktioniert RSA-Verschlüsselung?

2.1 Schlüsselpaare

Die RSA-Verschlüsselung ist ein sogenannter asymmetrischer Chiffre. Das bedeutet, daß zwei verschieden Schlüssel zum Ver- bzw. Entschlüsseln verwendet werden.

Beim RSA-Verfahren erhält jeder Teilnehmer zwei Schlüssel – das Paar aus privatem (Private Key) und öffentlichem (Public Key) Schlüssel. Man spricht auch von einem Public-Key-Verfahren.

Den privaten Schlüssel muß jeder Teilnehmer sehr sorgfältig aufbewahren, der öffentlichen dagegen kann bedenkenlos veröffentlicht werden.

2.2 Anwendung für Verschlüsselung

Will man nun einen Klartext verschlüsselt übermitteln, nimmt man den Public Key des Empfängers zum Verschlüsseln. Das RSA-Verfahren garantiert, daß sich der Private Key des Empfängers nicht (ohne immensen Aufwand) aus dem Public Key berechnen läßt. Weiterhin ist die verschlüsselte Nachricht *nur* mit dem Private Key des Empfängers entschlüsselbar (siehe auch Abb. 1).

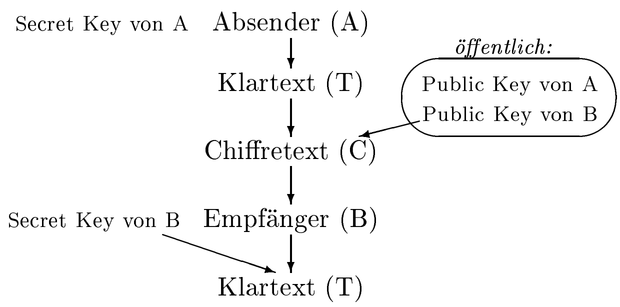

Abbildung 1: Anwendung der RSA-Verschlüsselung

2.3 Anwendung für Signieren

Ferner läßt sich das RSA-Verfahren verwenden, um Nachrichten elektronisch zu unterschreiben. Dabei wird (im Gegensatz zu herkömmlichen Unterschriften) auch die Unversehrtheit der empfangenen Daten gewährleistet.

Um einen Text zu signieren (siehe Abb. 2), wird zuerst mittels einer kryptografischen Funktion (sog. Hashfunktion) eine Art Quersumme über die Daten gebildet. Dieser Hash hat die Eigenschaft, daß man nicht mit vertretbarem Aufwand einen Text passend zu einem Hashwert konstruieren kann. Außerdem bewirken geringste Änderungen im Text große Änderungen im Hashwert.

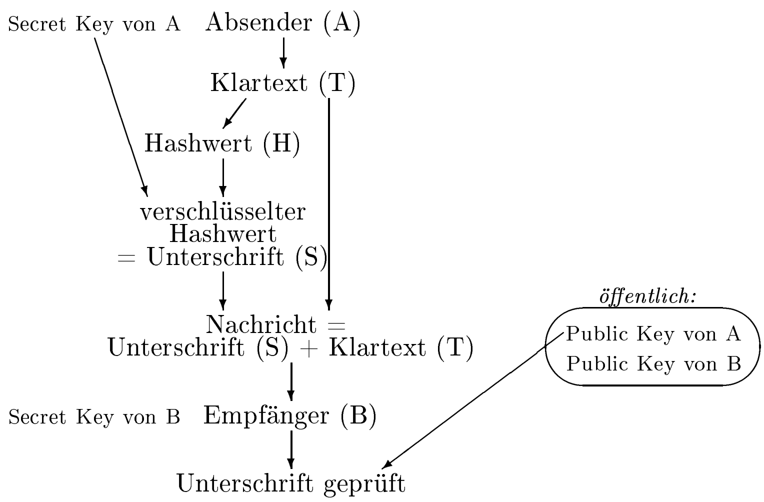

Abbildung 2: Anwendung von RSA zum Signieren

Dieser Hash-Wert wird dann mit dem privaten Schlüssel des Absenders verschlüsselt und der Nachricht begefügt.

Der Empfänger kann nun mit dem öffentlichen Schlüssel des Absenders den originalen Hashwert entschlüsseln und andererseits selbst den Hashwert für die empfangene Nachricht berechnen. Stimmen beide Werte überein, so ist die Nachricht in genau dem Zustand angekommen, in dem sie der Absender verschickt hat.

Zusätzlich ist damit auch „bewiesen“, daß die Nachricht vom vorgeblichen Absender stammt – jemand anderes hätte sie nicht mit dem privaten Schlüssel signieren können.

3 Arbeitsweise des `sshd` (aus der `man`-Page)

3.1 Ablauf des Verbindungsaufbaus

- Client baut TCP-Verbindung zum Server auf
- `sshd` schickt Host- und Server-Public-Keys zum Client
- Client:
  - checkt den Host-Key (Client hat eine Kopie des Public-Keys)
  - erzeugt 256 bit Zufallszahl
  - verschlüsselt Zufallszahl mit Host- und Server-Public-Keys
  - schickt verschlüsselte Zahl an Server
- ⇒ diese Zufallszahl ist dann der Session-Key für die Verschlüßelung der weiteren Kommunikation; nur der Besitzer der Host- und Server-Private-Keys konnte sie entschlüsseln; der Server-Key wird alle Stunde (nach Gebrauch) neu generiert
- Rest der Session wird verschlüßelt mit IDEA, DES, 3DES, ARCFOUR, TSS oder Blowfish
- danach erst erfolg die Authentifizierung des Nutzers

3.2 Authentifizierung

Client versucht nun, sich zu authentifizieren:

- mit `.rhosts` (unsicher!, per default abgeschaltet)
- mit `.rhosts` kombiniert mit RSA
- mit RSA challenge-response
- mit Passwort

⇒ `rshd`, `rlogind`, `rexecd` und `rexd` sollten deaktiviert sein!

3.3 `sshd`-Optionen

Eine Auswahl von Optionen, die der `sshd` versteht:

`-b bits`

Anzahl der Bits im Server-Key angeben (default: 768)

`-f configfile`

Konfigurationsdatei angeben (default: `/etc/sshd_config`)

`-h hostkeyfile`

Datei angeben, die den Host-Key enthält. (Standard: `/etc/ssh_host_key` Diese Datei ist normalerweise *nur* von root lesbar, denn sie enthält den privaten RSA-Schlueßel des Rechners.

`-i`

Gibt an, dass `sshd` vom `inetd` gestartet wurde

`-p port`

Port angeben, auf dem `sshd` lauscht (default: 22)

3.4 Datei `/etc/sshd_config`

Format: `Schlüsselwort Wert`

`AllowGroups`

Liste von Unix-Gruppen. Nutzer, der sich anmeldet, muß als primäre eine dieser Gruppen haben.

`AllowHosts`

Gibt (falls vorhanden) die exlusive Liste von Hosts an, die sich anmelden dürfen (default: alle Hosts; kann Wildcards `*` und `?` enthalten)

`AllowUsers`

Liste von Usernamen der Form `user` oder `user@host`, die zum Einloggen berechtigt sind. Die Einstellung wirkt zusätzlich zu anderen Restriktionen. (default: alle Nutzer; Wildcards möglich)

`DenyGroups`

Komplement zu `AllowGroups`

`DenyHosts`

Komplement zu `AllowHosts`

`DenyUsers`

Komplement zu `AllowUsers`

`HostKey`

Angabe der Host-Key-Datei (analog zu Option `-h`)

`IgnoreRhosts`

`rhosts` und `shosts` Dateien ignorieren für Authentifizierung; `/etc/hosts.equiv` und `/etc/shosts.equiv` werden weiterhin verwendet

`PasswordAuthentication`

Aktiviert Authentifizierung via Passwort (default: aktiviert)

`PermitRootLogin`

Logins von `root` erlauben (Ausführen von Kommandos als `root` ist trotzdem erlaubt)

`RhostsAuthentication`

Authentifizierung via `/etc/hosts.equiv` oder `/etc/rhosts` – sollte deaktiviert sein!

`RhostsRSAAuthentication`

wie oben, aber mit zusätzlicher RSA-Authentifizierung erlauben? (default: an)

`RSAAuthentication`

reine RSA-Authentifizierung aktivieren? (default: ja)

3.5 sonstige Dateien, die von `sshd` und `ssh` verwendet werden

`/etc/ssh_known_hosts` Diese Datei sollte vom Admin vorbereitet werden – sie sollte die öffentlichen Schlüssel aller Rechner enthalten, die per `ssh` Kontakt aufnehmen wollen.

Die Einträge entsprechen Kopien von `/etc/ssh_host_key.pub`.

`$HOME/.ssh/known_hosts` Diese Datei wird automatisch (interaktiv) generiert – jedesmal, wenn der Nutzer mit einem Host Kontakt aufnimmt, der noch unbekannt ist, wird er gefragt, ob der den Key akzeptieren will.

Die Einträge entsprechen Kopien von `/etc/ssh_host_key.pub`.

`/etc/rhosts.equiv` Diese Datei wird während der `.rhosts` Authentifizierung verwendet. Es enthält pro Zeile einen Hostnamen. Nutzer von diesen Hosts dürfen sich ohne Passwort anmelden, sofern sie den gleichen Usernamen auf beiden Maschinen verwenden.

Normalerweise ist zusätzlich eine erfolgreiche RSA Host-Authentifizierung notwendig! (Dazu unten mehr.)

Diese Datei wird zusätzlich von `rlogin` & Co. verwendet; sollte also mit größter Vorsicht behandelt werden, da die r-Tools unsicher sind!

`/etc/shosts.equiv` Äquivalent zu `/etc/rhosts.equiv`, wird aber nur von `sshd` verwendet, nicht von den r-Tools.

Diese Datei sollte vorzugsweise verwendet werden.

Zur Beachtung: Indem man diese oder ähnliche Dateien verwendet, wird die Sicherheit auf dem Server u. U. durch die Sicherheit auf dem Client beeinträchtigt. Ein Angreifer, der auf dem Client User-Rechte erlangt, kann diese dann unproblematisch auch auf dem Server erhalten!

4 RSA-Authentifizierung

RSA-Authentifizierung wird verwendet, um den Nutzer gegenüber dem Server zu authentifizieren, ohne daß dieser ein Passwort eingeben muß. Auf dem Server muß in `~/.ssh/authorized_keys` der öffentliche Schlüssel des Nutzers aufgelistet sein.

Der Server ist also im Besitz des öffentlichen Schlüssels, der Client (Nutzer) kennt sowohl den öffentlichen als auch den privaten Schlüssel.

- beim Anmelden sagt `ssh` bescheid, welcher Key verwendet werden soll
- der Server prüft, ob der Key erlaubt ist
- der Server erzeugt eine Challenge, eine Zufallszahl, verschlüsselt diese mit dem Public-Key des Clienten und schickt sie zum Client
- der Client besitzt als einziger den Private-Key, kann deshalb die verschlüsselte Challenge entschlüsseln ohne den Private-Key gegenüber dem Server preis zu geben.

Von der Client-Seite aus wurde bereits beim Verbindungsaufbau analog verfahren und somit die Authenzität des Servers geprüft, indem der Session-Key mit dem Public-Key des Servers verschlüsselt wurde.

Die Public/Private-Key-Paare werden mit `ssh-keygen` erzeugt und in `/etc/ssh_host_key` (private) und `/etc/ssh_host_key.pub` (public) gespeichert. Die Zugriffsrechte der Dateien werden automatisch entsprechend gesetzt (root-readonly bzw. world-readable, root-writeable).

5 Anwendung im konkreten Fall

Für die Realisierung wird folgendes Vorgehen vorgeschlagen:

- der Server und jeder Zielhost muß mittels ssh-keygen einen Host-Key erhalten:

  `ssh-keygen -f /etc/ssh_host_key -P ””`

  erzeugt einen Host-Key ohne Passphrase. Eventuell kann die Keylänge vergrößert werden mit der Option `-b`. Standardmäßig wird eine Keylänge von 1024 Bit verwendet. Mittlerweile sollte man eher auf 1536 oder 2048 Bit orientieren, obwohl der Rechenaufwand dadurch steigt.

- auf den Zielhosts wird in `/etc/shosts.equiv` der DNS-Name des Servers eingetragen

- in `/etc/ssh_known_hosts` der Clients wird der Public-Host-Key des Servers eingetragen und auf dem Server alle Public-Host-Keys der Clients

- für die Remote-Ausführung von Kommandos wird am Besten ein eigener User angelegt, in dessen Home-Verzeichnis

Es ist darauf zu achten, daß für alle Hostnamen die vollen Namen der Rechner verwendet werden und daß diese mit den Namen übereinstimmen, die per Reverse-DNS-Lookup der IPs erhalten werden. `ssh` prüft dies.

5.1 /etc/ssh_known_hosts

Die Datei enthält pro Zeile einen Host-Key. Dieser kann z. B. aus `/etc/ssh_host_key.pub` gewonnen werden, indem der Zeile noch der FQDN vorangestellt wird. (Eventuelle Aliases können dort auch gelistet werden, durch Komma getrennt.) Mit der kommerziellen `ssh` wird ein Perl-Skript `make-ssh-known-hosts` ausgeliefert, daß es ermöglicht, mit Hilfe der DNS-Records einer Domain alle Public-Keys zu erhalten und `/etc/ssh_known_hosts` zu erzeugen.

5.2 /etc/sshd_config

Die Standard-Konfiguration von `ssh` ist etwas zu restriktiv eingestellt - so ist `RhostsRSAAuthentication` standardmäßig deaktiviert. Weiterhin muß `IgnoreRHosts` auf „no“ eingestellt werden.

6 reale Anwendung von `ssh`

Jetzt sollen noch die Vorteile von `ssh` in der täglichen Arbeit erläutert werden:

- *Sichere* Verbindungen zu entfernten Rechnern. Eine `telnet`-Verbindung kann mit einfachsten Mitteln übernommen werden.
- X11-Forwarding: Arbeitet man unter X, wird durch `ssh` standardmäßig die X11-Verbindung weitergeleitet. Man braucht nicht mit

  ```
  xhost +
  ```

  oder Ähnlichem seine Rechnersicherheit zu kompromittieren oder X-Cookies von Hand zu kopieren. Nebenbei wird die X-Verbindung verschlüsselt.
- Sicherer Filetransfer mit `scp`:

  ```
  $ scp [-C][-r] [user@][host:]file [user@][host:]file
  ```

- Kompression der Verbindung: Durch Angabe der Option `-C` wird die Verbindung mit dem `gzip`-Algorithmus komprimiert.
- Port-Forwarding: Man kann mit `ssh` vom lokalen Host zu einem Host auf der „anderen Seite" einen TCP/IP-Tunnel aufbauen.

  Zum Beispiel gibt es an der TU einen Rechner clic0a2.hrz.tu-chemnitz.de, der von außerhalb des Universitätsnetzes nicht erreichbar ist, obwohl er eine offizielle IP hat. Um auf den Webserver auf diesem Host zugreifen zu können, reicht es, eine `ssh` auf einen Compute-Server der Uni zu machen und dabei ein Port-Forwarding zu aktivieren:

  ```
  $ ssh odoaker.hrz.[...].de -L 4242:clic0a2.hrz.[...].de:80
  ```

  Damit leitet man den lokalen Port 4242 auf den Port 80 des Rechner clic0a2.hrz um. Der Rechner odoaker dient dabei nur als „Einsprungpunkt" in das entfernte Netz.

  Dieses Vorgehen funktioniert auch in umgekehrter Richtung - ein Port auf dem Zielrechner kann auch auf einen Rechner und Port im lokalen Netz umgeleitet werden; dazu dient dann die Option `-R`.
- Extra-Sicherheit für RSA-Authentifizierung, indem man seine „Identity" (`~/.ssh/identity`) mit einer Passphrase versieht und mit Hilfe des `ssh-agent` nach einmaliger Eingabe der Passphrase auf alle per RSA-Authentifizierung erreichbaren Hosts Zugriff erhält. Dies verhindert zuverlässig, daß ein Administrator mit `root`-Rechten die eigene Identity verwenden kann, um Zugriff auf entfernte Rechner zu erhalten. Weiterhin kann man damit mehrere Identitäten verwalten, z. B. eine für das lokale Netz, und jeweils eine pro Firmenkunde zur Fernwartung.

7 Der ssh-agent

Mit dem `ssh-agent` läßt sich eine sehr komfortable und trotzdem sichere Authentifizierungsinfrastruktur aufbauen. Ab S.u.S.E. 6.4 genügt es, in der Datei `~/.xsession` die Variable `usessh` auf „yes" zu stellen und der `ssh-agent` wird automatisch beim Starten von X ausgeführt.

Über eine Umgebungsvariable teilt der Agent den anderen `ssh`-Programmen mit, wo er zu finden ist; diese sprechen ihn dann bei Bedarf automatisch an.

Die Aufgabe des Agenten ist es, private Identitäten zu verwalten. Er ermöglicht es, eine Identität mit einer Passphrase zu versehen ohne daß man diese bei jeder Verwendung der Identität erneut eingeben muß.

Mit dem Kommando `ssh-add` fügt man eine neue Identität zum Agenten hinzu (und wird bei dieser Gelegenheit nach der Passphrase gefragt, falls nötig).

Ist die Identität dem Agenten bekannt, wird sie automatisch von der `ssh` zur Authentifizierung benutzt.

Weiteren Komfort gewinnt man dadurch, daß die Verbindung zum `ssh-agent` automatisch durch den `ssh`-Kanal weitergeleitet wird und somit auch auf dem entfernten Rechner zur Verfügung steht.

7.1 Anwendung im Detail

Diese Infrastruktur läßt sich zum Beispiel verwenden, um die Zugänge zur Remote-Administration von Kundenrechnern zu sichern.

Dazu generiert man für jeden Kunden (bzw. für jede Sicherheitszone) ein eigenes Keypaar mit dem Kommando `ssh-keygen` und versieht dieses mit einer Passphrase. Das Keypaar kann im eigenen Firmennetz an zentraler Stelle aufbewahrt werden – ohne Passphrase ist es nutzlos.

Beim Kunden wiederum wird in der Datei `~/.ssh/authorized_keys` eine Kopie des öffentlichen Teils des Keypaars hinterlegt und ermöglicht somit den Zugang. (Die Konfiguration des `sshd` sollte überprüft werden; sie muß RSA-Authentifizierung zulassen.)

Will man nun auf den Kundenrechner zugreifen, fügt man mit

```
$ ssh-add /global/security/customerkeys/ein_kunde
```

die private Identität zum Agenten hinzu und kann nach Eingabe der Passphrase mit `ssh` direkt auf dem entfernten Rechner arbeiten. Dabei stellt eine Firewall keine Grenze dar – man sollte dann zwei Identitäten erzeugen: eine für die Firewall und eine zweite für den internen Kundenserver.

Weiterer Nebeneffekt: Man kann explizite (Remote-)Wartungszugänge einrichten, die lokal nicht verwendet werden können, da sie kein Login zulassen (ungültiger Passworthash in `/etc/shadow`).

8 Weiterführende Informationen

Umfangreiche deutsche Dokumentatin findet sich zum Beispiel auf den Seiten von Holger Trapp:

```
http://www.tu-chemnitz.de/~hot/ssh/
```