SSH – Secure Shell

Die Secure Shell im Einsatz
„LugBE Off-Event Vortrag“

Patrik Schilt <patrik@schilt.ch>
22. Januar 2004
Restaurant Beaulieu, Bern

Einführung in Kryptographie

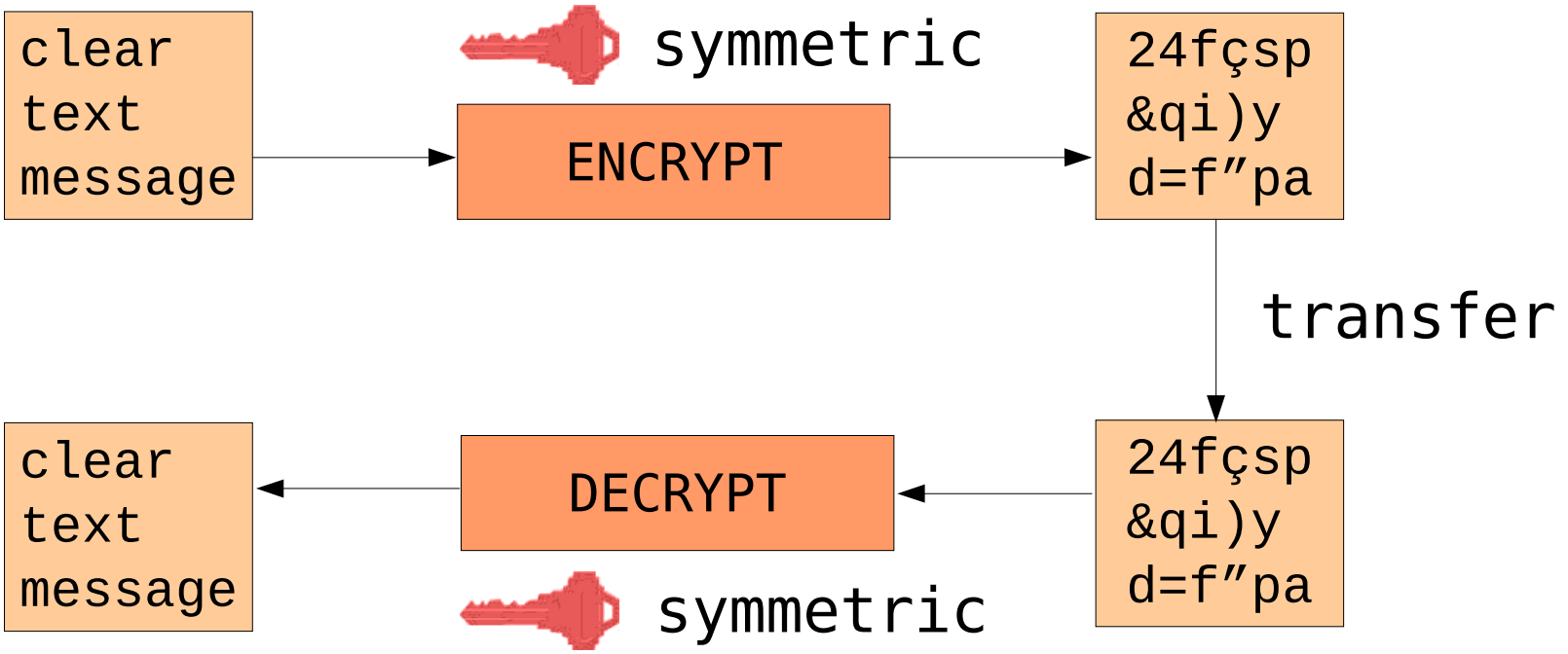
Um SSH oder andere kryptographische Applikationen wie SSL (Secure Sockets Layer) zu verstehen benötigen wir ein Verständnis für

- **Kryptographische Algorithmen**
- **Message Digest Funktionen** (Hash Funktionen)
- **Digitale Signaturen**

Diese Techniken bilden die Grundlage für die Privatsphäre, Integrität und Authentisierung in der digitalen Welt.

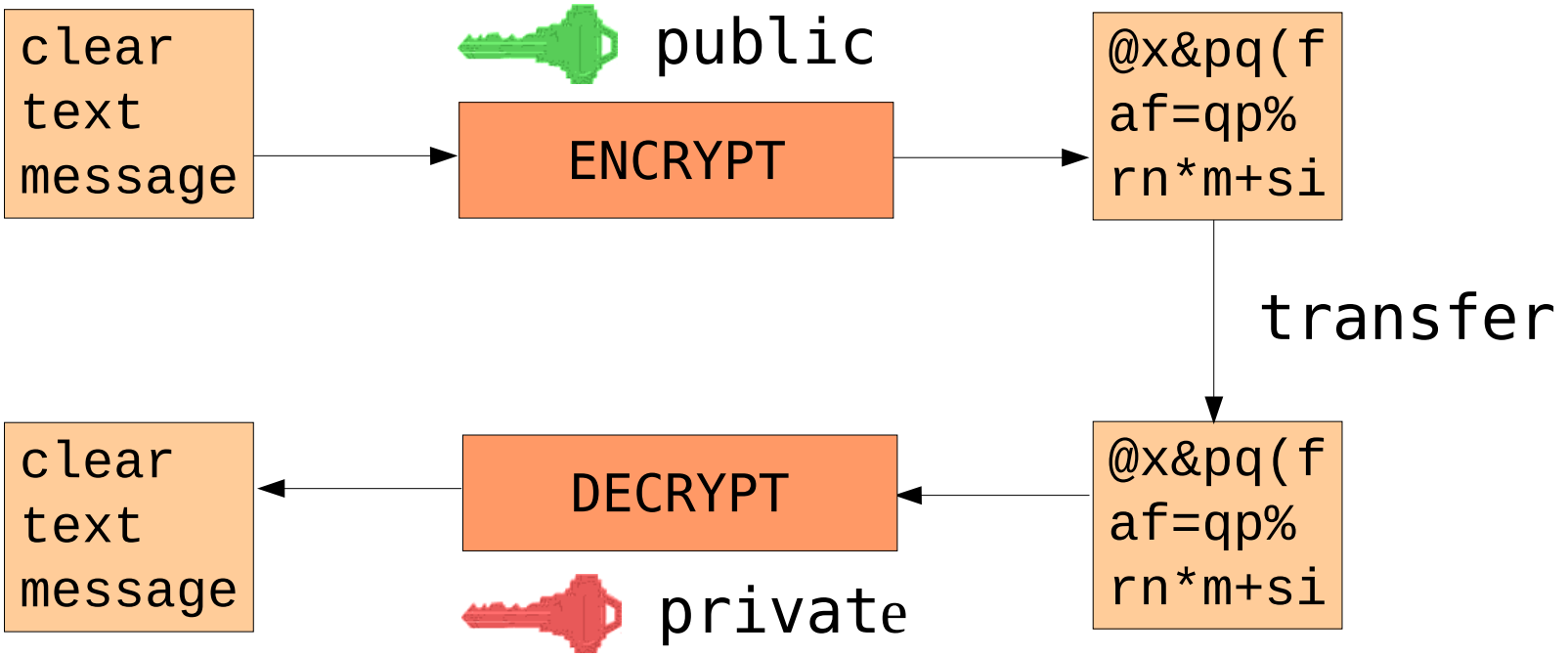
Einführung in Kryptographie

Konventionelle Kryptographie, auch bekannt als **symmetrische** Kryptographie, basiert darauf, dass der Sender und der Empfänger **einen geheimen Schlüssel** verwenden um die Nachricht zu verschlüsseln, respektive zu entschlüsseln. Der Austausch dieses Schlüssels kann jedoch recht problematisch sein. (Die bekannten Verschlüsselungsverfahren sind AES, 3DES und Blowfish)



Einführung in Kryptographie

Public key Kryptographie, auch bekannt als **asymmetrische** Kryptographie, löst das Problem des Schlüsselaustauschs indem ein neues Verfahren gewählt wurde, welches zwei Schlüssel verwendet. Wurde die Nachricht mit dem einen Schlüssel verschlüsselt, so kann diese nur mit dem zweiten Schlüssel wieder entschlüsselt werden. Es kann also sicher kommuniziert werden indem beide Kommunikationspartner einfach einen Schlüssel publizieren (**public key**) und den anderen gut aufbewahren (**private key**). Der Sender verschlüsselt dann die Nachricht mit dem public key und der Empfänger entschlüsselt diese mit seinem private key. (Die bekannten Verfahren sind RSA, DSA und ElGamal)



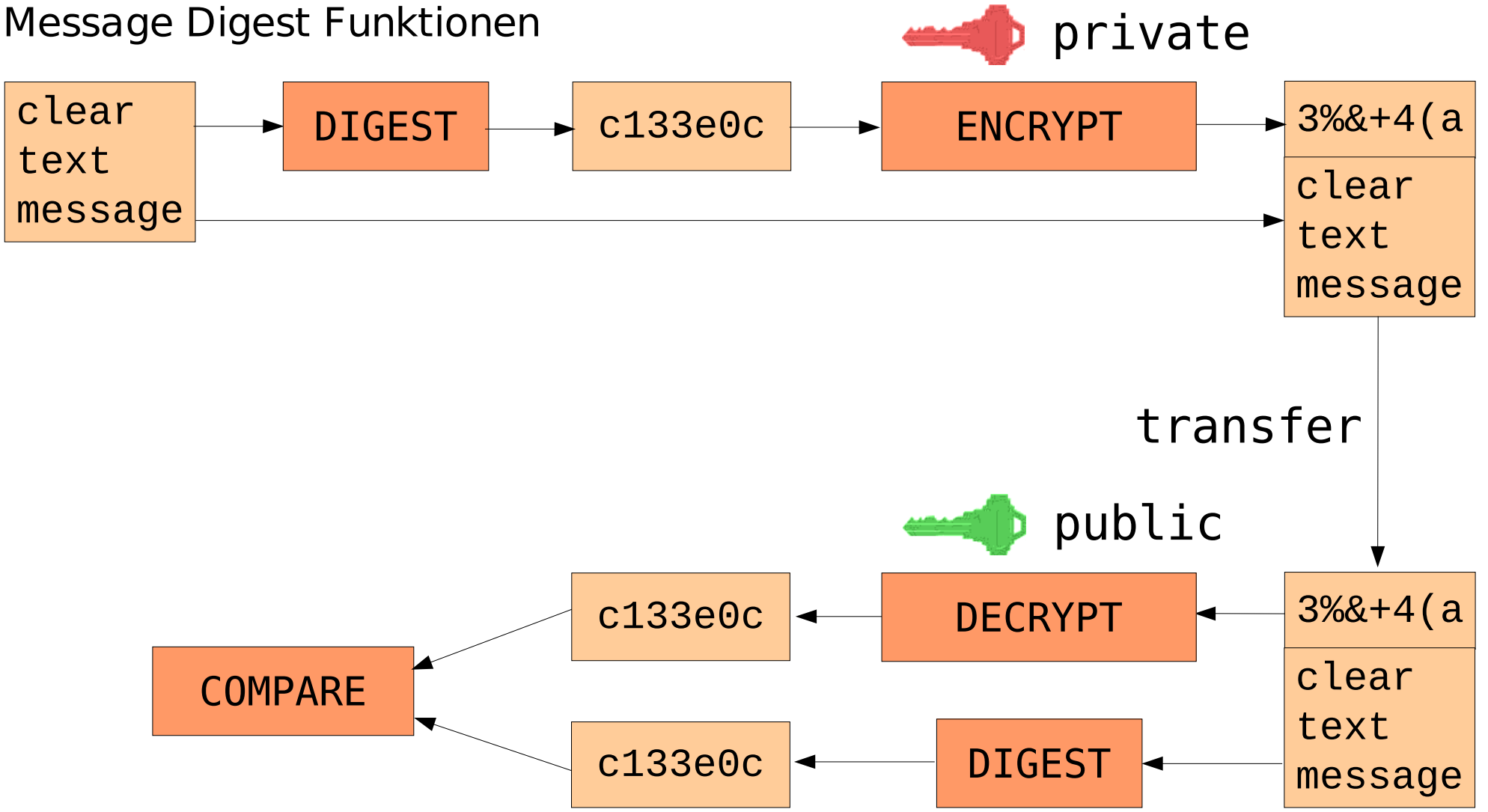
Einführung in Kryptographie

Eine **Message Digest Funktion**, auch als Hash Funktion bekannt ist eine Einwegfunktion. Diese werden dazu verwendet, eine kurze immer gleich lange Zeichenfolge von verschiedenen Nachrichten zu generieren. Digest Algorithmen wurden so entwickelt, dass es nicht möglich ist von dem Digest auf die Nachricht zu schliessen, und dass zwei verschiedene Nachrichten nicht den gleichen Digest ergeben. (Die bekannten Verfahren sind MD5 und SHA-1)

Digitale Signaturen werden kreiert, indem der Sender den Digest einer Nachricht mit seinem private key verschlüsselt, man spricht dann von signieren. Jeder kann die Nachricht lesen und mit dem public key verifizieren. Stimmt der Digest überein, stammt die Nachricht tatsächlich vom Sender und wurde während der Übertragung nicht verändert. (Die bekannten Verfahren sind RSA und DSS)

Einführung in Kryptographie

Digitale Signaturen und
Message Digest Funktionen



Secure Shell – SSH und OpenSSH

- SSH (Secure Shell) ermöglicht eine sichere stark verschlüsselte Kommunikation zwischen zwei Hosts über ein unsicheres Netzwerk.

- Das SSH-1 Protokoll sowie die erste SSH Software wurde 1995 von Tatu Ylönen entwickelt, einem Forscher der Helsinki University of Technology in Finland. SSH1 wurde als freie Software publiziert.

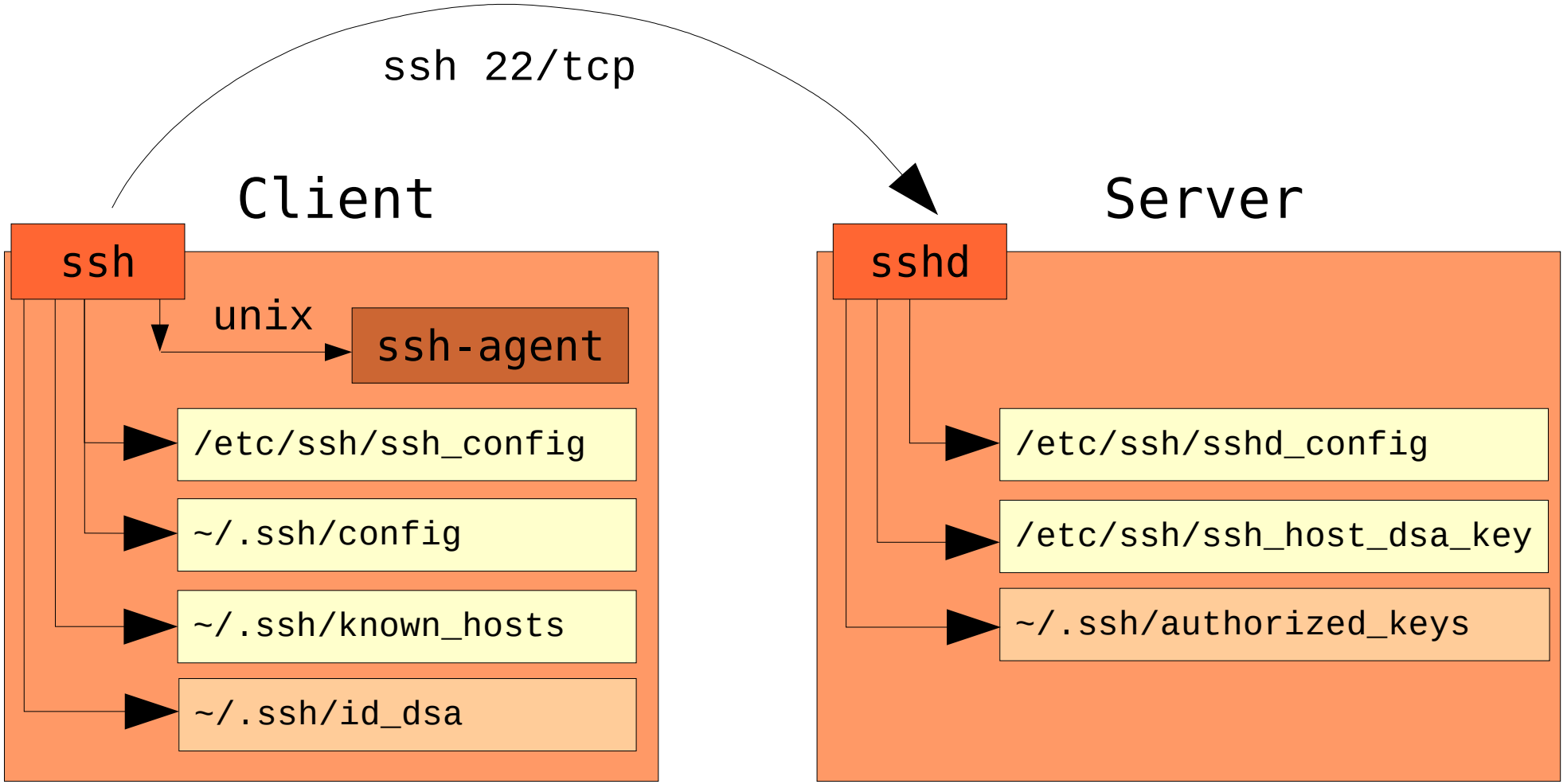
- SSH-2.0 wurde 1997 als Internet Draft durch eine Arbeitsgruppe (secsh) der IETF publiziert und SSH wird weiter von dieser Gruppe standartisiert. http://www.ietf.org/html.charters/secsh-charter.html

- OpenSSH ist eine **freie** Implementation des SSH Protokolls und geniesst eine weite Verbreitung. OpenSSH stammt aus dem OpenBSD Projekt und wird auch von diesem stetig weiterentwickelt. http://www.openssh.org

- OpenSSH beinhaltet die Programme **ssh**, als Ersatz für rlogin, rsh und telnet, **scp** als Ersatz für rcp, und **sftp** welches ftp ersetzen kann. Weiter ist der **sshd** wichtig, welcher den Server Teil der Applikation ausmacht. Weiter kommen die Hilfsprogramme **ssh-add**, **ssh-agent**, **ssh-keygen** und andere dazu. OpenSSH unterstützt die SSH Protokoll Versionen 1.3, 1.5, und 2.0, wobei Version 2.0 zu empfehlen ist.

Secure Shell - OpenSSH

- Grundprinzip und die wichtigsten Dateien



OpenSSH - ssh(1), scp(1), sftp(1)

Beispiele (Basisanwendung)

Wir möchten eine Verbindung mit einem entfernten Rechner aufbauen und dabei einen bestimmten Login Benutzer verwenden.

```
$ ssh -l schilt www.lugbe.ch
```

Können wir einmal nicht einloggen, kann uns die Option -v eventuell weiterhelfen.

```
$ ssh -v -l schilt www.lugbe.ch
OpenSSH_3.6.1p2, SSH protocols 1.5/2.0, OpenSSL 0x0090702f
debug1: Reading configuration data /home/patrik/.ssh/config
debug1: Reading configuration data /etc/ssh/ssh_config
...
```

Eine Datei auf den entfernten Rechner kopieren.

```
$ scp lugbe-ssh-vortrag.sxi schilt@www.lugbe.ch:
schilt@www.lugbe.ch's password: <PASSWORD>
lugbe-ssh-vortrag.sxi                         100%   32KB   1.2MB/s   00:00
```

sftp kann wie das klassische ftp verwendet werden.

```
$ sftp schilt@www.lugbe.ch
Connecting to www.lugbe.ch...
schilt@www.lugbe.ch's password: <PASSWORD>
sftp> get lugbe-ssh-vortrag.sxi
lugbe-ssh-vortrag.sxi                         100%   32KB   1.2MB/s   00:00
sftp> quit
```

OpenSSH - ssh(1), ssh-keygen(1)

Beispiele (Host Keys)

Anzeigen der DSA (Digital Signature Algorithm) Host Fingerprints.

```
kalinka$ ssh-keygen -l -f /etc/ssh/ssh_host_dsa_key.pub
1024 cd:78:b1:fc:c6:45:6a:be:64:5d:21:9f:77:3f:2d:13 /etc/ssh/ssh_host_dsa_key.pub
```

Bei der ersten Verbindung mit einem neuen Rechner speichert der SSH Client jeweils den Public Host Key in **~/.ssh/known_hosts** ab, und vergleicht diesen bei jeder weiteren Verbindung, damit wir auch sicher sind, dass wir mit dem richtigen Rechner kommunizieren. Wir sollten aber das erste mal besonders vorsichtig sein und den Fingerprint verifizieren.

```
$ ssh -l schilt www.lugbe.ch
The authenticity of host 'www.lugbe.ch (212.103.66.68)' can't be established.
DSA key fingerprint is cd:78:b1:fc:c6:45:6a:be:64:5d:21:9f:77:3f:2d:13.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'www.lugbe.ch' (DSA) to the list of known hosts.
```

DSA Host Key später überprüfen.

```
$ ssh-keygen -l -f ~/.ssh/known_hosts | grep lugbe
1024 cd:78:b1:fc:c6:45:6a:be:64:5d:21:9f:77:3f:2d:13 www.lugbe.ch
```

OpenSSH - ssh(1), ssh-keygen(1)

Beispiele (Man-in-the-middle)

Falls folgende Ausgabe erscheint hat der Vergleich der public Host Keys fehlgeschlagen und wir sollten kurz nachdenken. Häufig wurde einfach nur der Server neu installiert, oder die Keys wurden neu generiert, aus welchem Grund auch immer. Aber es könnte auch sein, dass wir Opfer einer man-in-the-middle Attacke geworden sind.

```
$ ssh -l schilt www.lugbe.ch
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the DSA host key has just been changed.
The fingerprint for the DSA key sent by the remote host is
cd:78:b1:fc:c6:45:6a:be:64:5d:21:9f:77:3f:2d:13.
Please contact your system administrator.
Add correct host key in /home/patrik/.ssh/known_hosts to get rid of this message.
Offending key in /home/patrik/.ssh/known_hosts:22
DSA host key for www.lugbe.ch has changed and you have requested strict checking.
Host key verification failed.
```

OpenSSH - ssh(1), ssh-keygen(1)

Beispiele (Public Key Authentisierung)

Ein neues DSA key Paar für die Benutzerauthentisierung erstellen.

```
$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/patrik/.ssh/id_dsa): <ENTER>
Created directory '/home/patrik/.ssh'.
Enter passphrase (empty for no passphrase): <ENTER>
Enter same passphrase again: <ENTER>
Your identification has been saved in /home/patrik/.ssh/id_dsa.
Your public key has been saved in /home/patrik/.ssh/id_dsa.pub.
The key fingerprint is:
e9:7b:96:20:1a:11:db:7d:3d:04:bf:00:b9:e4:f3:83 patrik@localhost
$ ls -l
total 8
-rw-------    1 patrik   patrik        668 Jan 18 13:25 id_dsa
-rw-r--r--    1 patrik   patrik        606 Jan 18 13:25 id_dsa.pub
```

Dann kopieren wir den public key auf den entfernten Server, so dass wir auf diesen Server via public/private key Authentisierung einloggen können.

```
$ cat id_dsa.pub | ssh -l schilt www.lugbe.ch cat ">>" .ssh/authorized_keys
schilt@www.lugbe.ch's password: <PASSWORD>
```

OpenSSH - ssh(1), ssh-keygen(1)

Beispiele (Public Key Authentisierung)

Wurde die „authorized_keys“ Datei neu erstellt, muss die Berechtigung richtig gesetzt werden, sonst wird es nicht funktionieren.

```
$ ssh -l schilt www.lugbe.ch chmod 600 .ssh/authorized_keys
```

Nun können wir via public/private Key Authentisierung auf den entfernten Rechner zugreifen. Sogar ohne ein Passwort einzugben!

```
$ ssh -l schilt www.lugbe.ch
kalinka$
```

Es ist sinnvoll den private key durch ein Passwort zu schützen. Dies kann jederzeit wieder geändert werden.

```
$ ssh-keygen -p -f .ssh/id_dsa
Key has comment '.ssh/id_dsa'
Enter new passphrase (empty for no passphrase): <NEW PASSWORD>
Enter same passphrase again: <NEW PASSWORD>
Your identification has been saved with the new passphrase.
```

Von nun an muss immer zuerst ein Passwort eingegeben werden bevor der SSH Client den private key verwenden kann. Auf einem Laptop sollten wir die private keys nur Passwort geschützt ablegen!

```
$ ssh -l schilt www.lugbe.ch
Enter passphrase for key '/home/patrik/.ssh/id_dsa': <NEW PASSWORD>
```

OpenSSH - ssh-agent(1), ssh-add(1)

Beispiele (Authentisierungs Agent)

Private keys ohne Passwort abzuspeichern kann ein Risiko darstellen, aber es ist mühsam immer wieder das gut gewählte Passwort einzugeben. Deshalb haben wir die Möglichkeit einen Agenten einzusetzen, welcher für uns die Schlüssel im Arbeitsspeicher hält. Wir können beliebig viele Schlüssel dem Agenten hinzufügen und der SSH Client greift für die Authentisierung einfach auf den Agenten zurück und eine Passwortabfrage ist nur beim Hinzufügen der Schlüssel nötig. Der SSH Client kommuniziert mit dem Agenten via Unix Domain Sockets.

Starten wir den Agenten, setzen die nötigen Umgebungsvariablen und lassen die Schlüssel auflisten. Ist vorerst natürlich noch kein Schlüssel vorhanden.

```
$ ssh-agent > .ssh-agent-environment
$ cat .ssh-agent-environment
SSH_AUTH_SOCK=/tmp/ssh-gszD3120/agent.3120; export SSH_AUTH_SOCK;
SSH_AGENT_PID=3121; export SSH_AGENT_PID;
echo Agent pid 3121;
$ . .ssh-agent-environment
Agent pid 3121
$ ssh-add -l
The agent has no identities.
```

Der Authentisierungs Agent kann auch wieder ganz einfach gestoppt werden.

```
$ ssh-agent -k
```

OpenSSH - ssh-agent(1), ssh-add(1)

Beispiele (Authentisierungs Agent)

Laden wir nun unseren bereits erstellten private key in den Agenten. Wenn wir uns wieder mit dem entfernten Rechner verbinden begegnen wir keiner Passwortabfrage obwohl der Schlüssel im Dateisystem mit einem Passwort geschützt ist. Dies deshalb weil der SSH Client den Agenten benutzt.

```
$ ssh-add .ssh/id_dsa
Enter passphrase for .ssh/id_dsa: <PASSWORD>
Identity added: .ssh/id_dsa (.ssh/id_dsa)
$ ssh-add -l
1024 e9:7b:96:20:1a:11:db:7d:3d:04:bf:00:b9:e4:f3:83 .ssh/id_dsa (DSA)
$ ssh -l schilt www.lugbe.ch
kalinka$ exit
```

Mit Hilfe der Option **ForwardAgent (-A)** weisen wir den SSH Client an, dass der entfernte Rechner auf unseren lokalen Agenten zurückgreifen kann. Wir können so von Host zu Host springen ohne je ein Passwort einzugeben und ohne dass dabei der private Schlüssel weitergereicht werden muss.

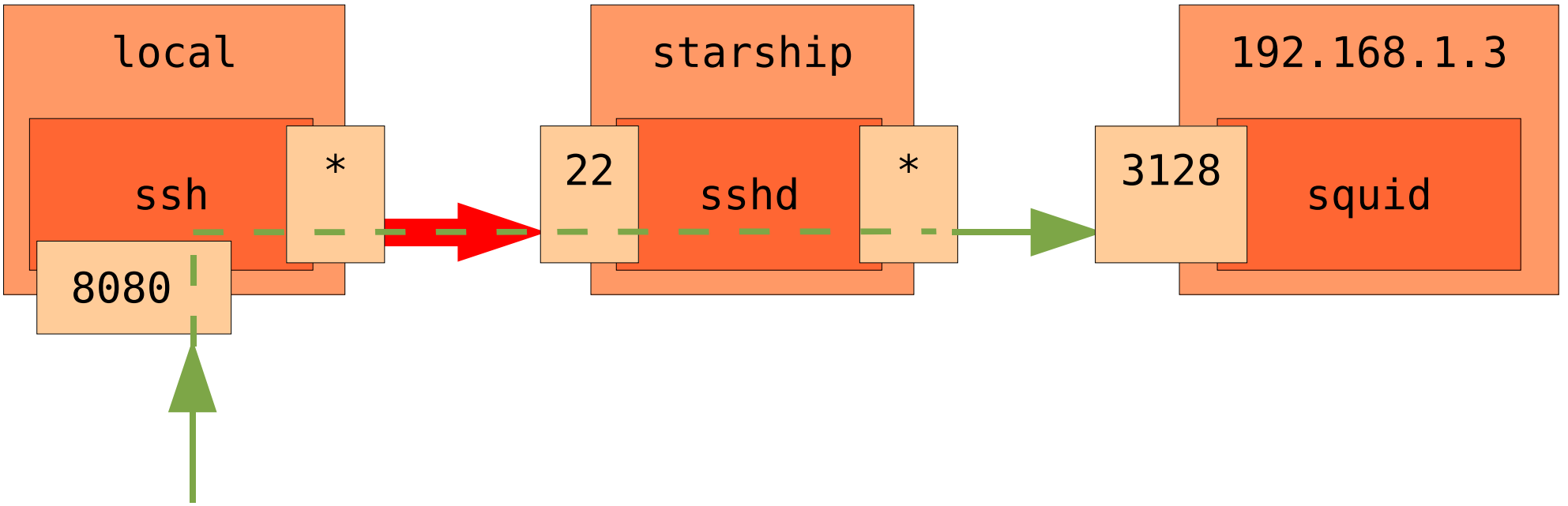
```
$ ssh-add -l
1024 e9:7b:96:20:1a:11:db:7d:3d:04:bf:00:b9:e4:f3:83 .ssh/id_dsa (DSA)
$ ssh -A quickstar
quickstar$ ssh-add -l
1024 e9:7b:96:20:1a:11:db:7d:3d:04:bf:00:b9:e4:f3:83 .ssh/id_dsa (DSA)
quickstar$ $ ssh -l schilt www.lugbe.ch
kalinka$ exit
```

OpenSSH - ssh(1)

Beispiele (Port Forwarding)

Mit der Option **LocalForward (-L)** leitet SSH die Anfragen auf einen bestimmten lokalen Port (8080) weiter auf einen gewünschten Zielrechner (192.168.1.3) mit dem gewünschten Port (3128). Der Zielrechner (192.168.1.3) muss natürlich vom Login Host (starship) aus erreichbar sein. In diesem Beispiel läuft auf 192.168.1.3 ein Squid HTTP Proxy.

```
$ ssh -L 8080:192.168.1.3:3128 starship
starship$
```

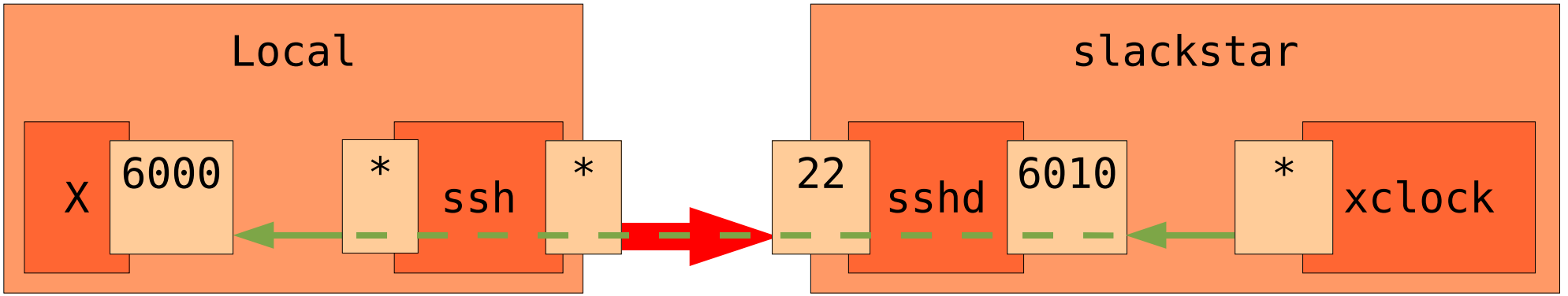


OpenSSH - ssh(1)

Beispiele (X11 Forwarding)

Mit der Option **ForwardX11 (-X)** werden X11 Verbindungen automatisch über die SSH Verbindung umgeleitet und die Umgebungsvariable DISPLAY wird korrekt gesetzt.

```
$ echo $DISPLAY
:0.0
$ netstat -an | grep 6000
tcp        0      0 0.0.0.0:6000            0.0.0.0:*               LISTEN
$ ssh -X slackstar
slackstar$ echo $DISPLAY
localhost:10.0
slackstar$ netstat --ip -an | grep 6010
tcp        0      0 127.0.0.1:6010          0.0.0.0:*               LISTEN
slackstar$ xclock &
```



OpenSSH - ssh_config(1)

Beispiele (~/.ssh/config, /etc/ssh/ssh_config)

```
#### Globale Konfigurationen ####
# Wir bevorzugen den DSS Algorithmus für die Digitale Signatur.
HostKeyAlgorithms ssh-dss,ssh-rsa
# Kompression zeichnet sich vorallem beim kopieren mit scp(1) aus.
Compression yes
# SSH Version 2 ist auf jeden Fall die erste Wahl!
Protocol 2,1
# Blowfish ist mein persönlicher Favorit.
Ciphers blowfish-cbc,3des-cbc

#### Host spezifische Konfigurationen ####
Host lugbe
        HostName www.lugbe.ch
        IdentityFile ~/.ssh/www.lugbe.ch/id_dsa
        User schilt

Host starschip
        HostName starship.schilt.ch
        User patrik
        ForwardAgent
        IdentityFile ~/.ssh/id_dsa
        LocalForward 8080 192.168.1.3:80
```

OpenSSH - sshd_config(1)

Beispiele (/etc/ssh/sshd_config)

Die Default Werte der SSH Daemon Konfigurationsdatei /etc/ssh/sshd_config wurden so gewählt, dass OpenSSH sofort eingesetzt werden kann, ohne dass dabei ein grosses Sicherheitsrisiko besteht. Oft möchte man aber den Daemon weiter einschränken, wie zum Beispiel im folgenden Beispiel.

```
# Protokoll Version 1 nicht mehr zulassen
Protocol 2

# root darf nicht einloggen, nur ein nicht-privilegierter admin
PermitRootLogin no
AllowUsers admin

# RSA Authentisierung nicht erlauben (betrifft nur Version 1)
RSAAuthentication no

# Passwort/PAM Authentisierung nicht erlauben
PasswordAuthentication no
PermitEmptyPasswords no
ChallengeResponseAuthentication no

# X11 Forwarding nicht erlauben
X11Forwarding no
```

OpenSSH - Tips

Tips

1) **Überprüfe immer die Fingerprints!** Die Tools dsniff und ettercap zeigen dass eine Man-in-the-middle Attacke einfach realisiert werden kann.
Dsniff - http://www.monkey.org/~dugsong/dsniff
Ettercap - http://ettercap.sourceforge.net

2) Verwende keychain(1) von Daniel Robbins um den ssh-agent(1) komfortabel einzusetzen. http://www.gentoo.org/proj/en/keychain.xml

3) Denke daran, dass SSH die Funktionalität der RSH (Remote Shell) komplett abdeckt und somit auch sehr gut mit Pipes umgehen kann.

Kopiere zum Beispiel Dateien eines lokalen Quellverzeichnises (SOURCEDIR) auf einen entfernten Rechner (SERVER) in ein bestimmtes Zielverzeichnis (DESTDIR).

```
$ (cd SOURCEDIR && tar cf - . ) | ssh SERVER "(cd DESTDIR && tar xvpf - )"
```

Oder kreiere ein Backup des Bootsektors.

```
# dd if=/dev/hda bs=512 count=1 | ssh SERVER dd of=bootsector.bin
1+0 records in
1+0 records out
1+0 records in
1+0 records out
```

Ende

http://www.openssh.com