Secure Shell (ssh)

Thorsten Bormer

27.01.2006

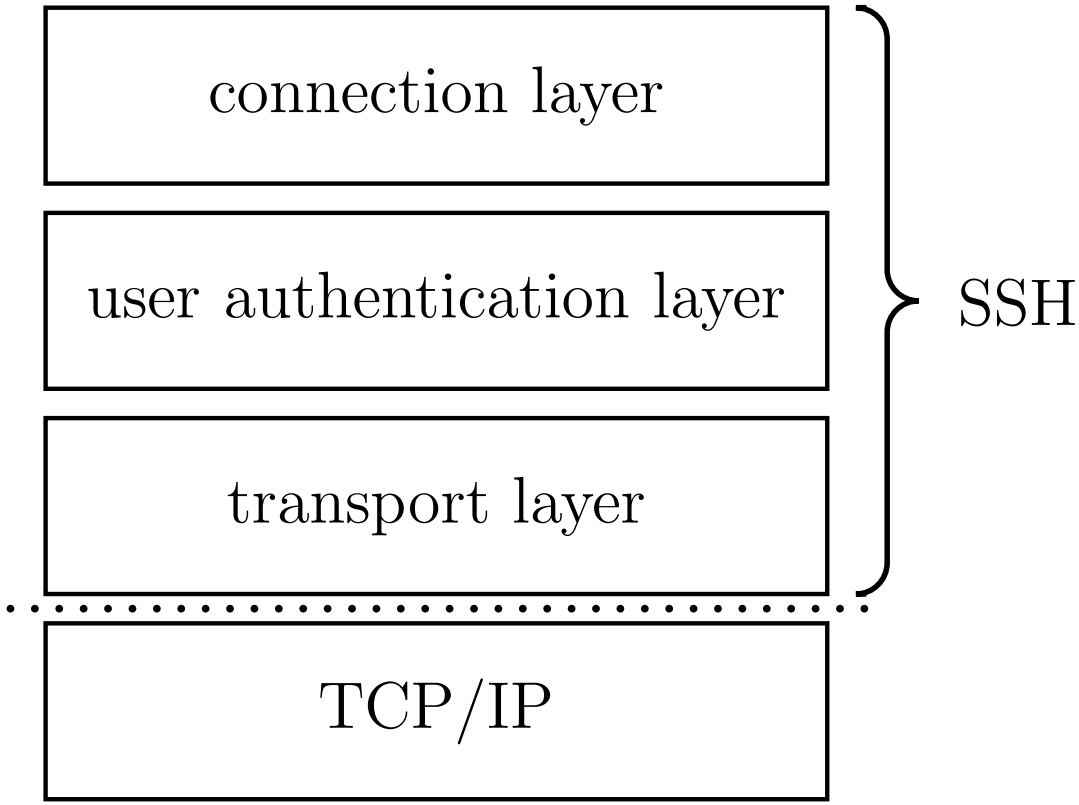
Was ist SSH?

- ssh bezeichnet sowohl ein Protokoll, als auch eine Menge von Programmen.
- Ziel: eine sichere Verbindung zwischen zwei Rechnern über „unsichere“ Leitung.
- ssh-tools Ersatz für:
  telnet, rlogin, rsh $\rightsquigarrow$ ssh (remote login)
  rcp, ftp $\rightsquigarrow$ scp, sftp (remote file transfer)

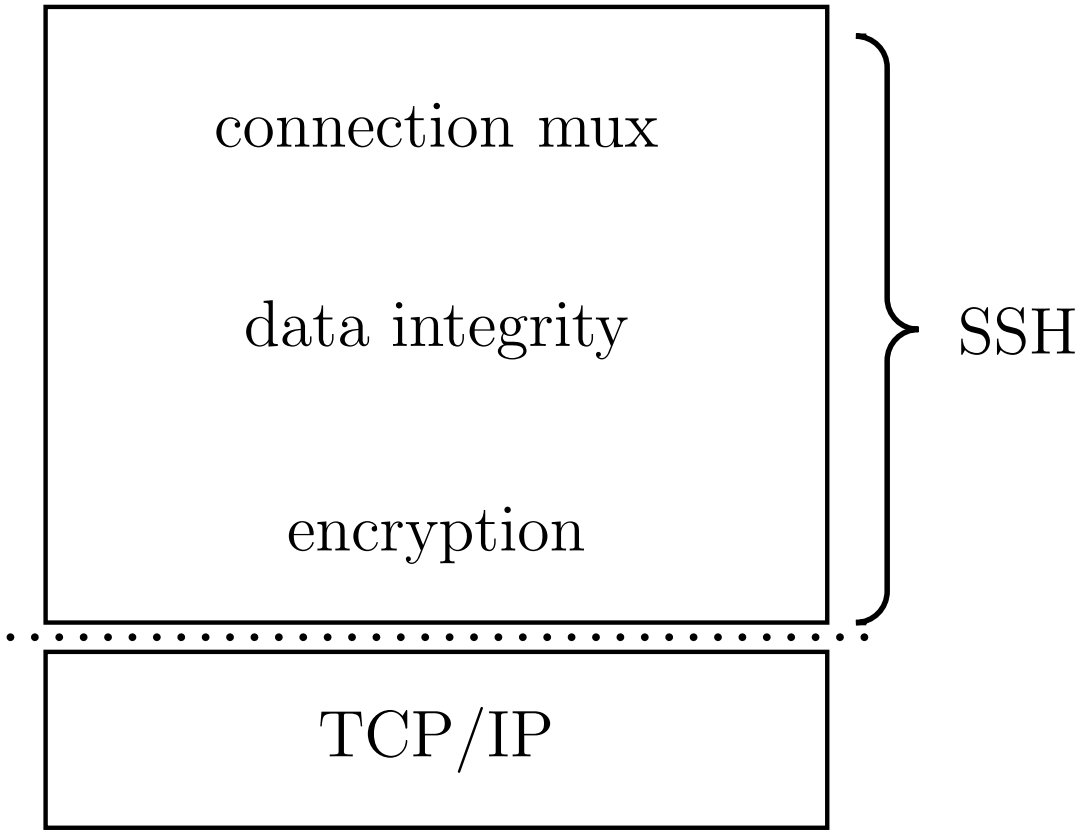
Entwicklung von SSH

- 1971, Telnet (RFC 139)
- 1983, Sep U.S. Patent 4,405,829 ('RSA'-Patent)
- 1991, Jul U.S. Patent 5,231,668 ('DSA'-Patent)
- 1995 SSH-1 (Tatu Ylönen)
- 1995, Jul. erste Implementation von Ylönen, freeware
- 1995, Dez. ssh Version 1.2.12
- 1996 SSH-2
- 1999 OSSH (Björn Grönvall)
- 1999 OpenSSH, verfügbar mit OpenBSD 2.6

Aufbau des Protokolls (SSH2)



Aufbau des Protokolls (SSH1)



Vereinbarungen

- SSH, Version 2
- Implementation: OpenSSH

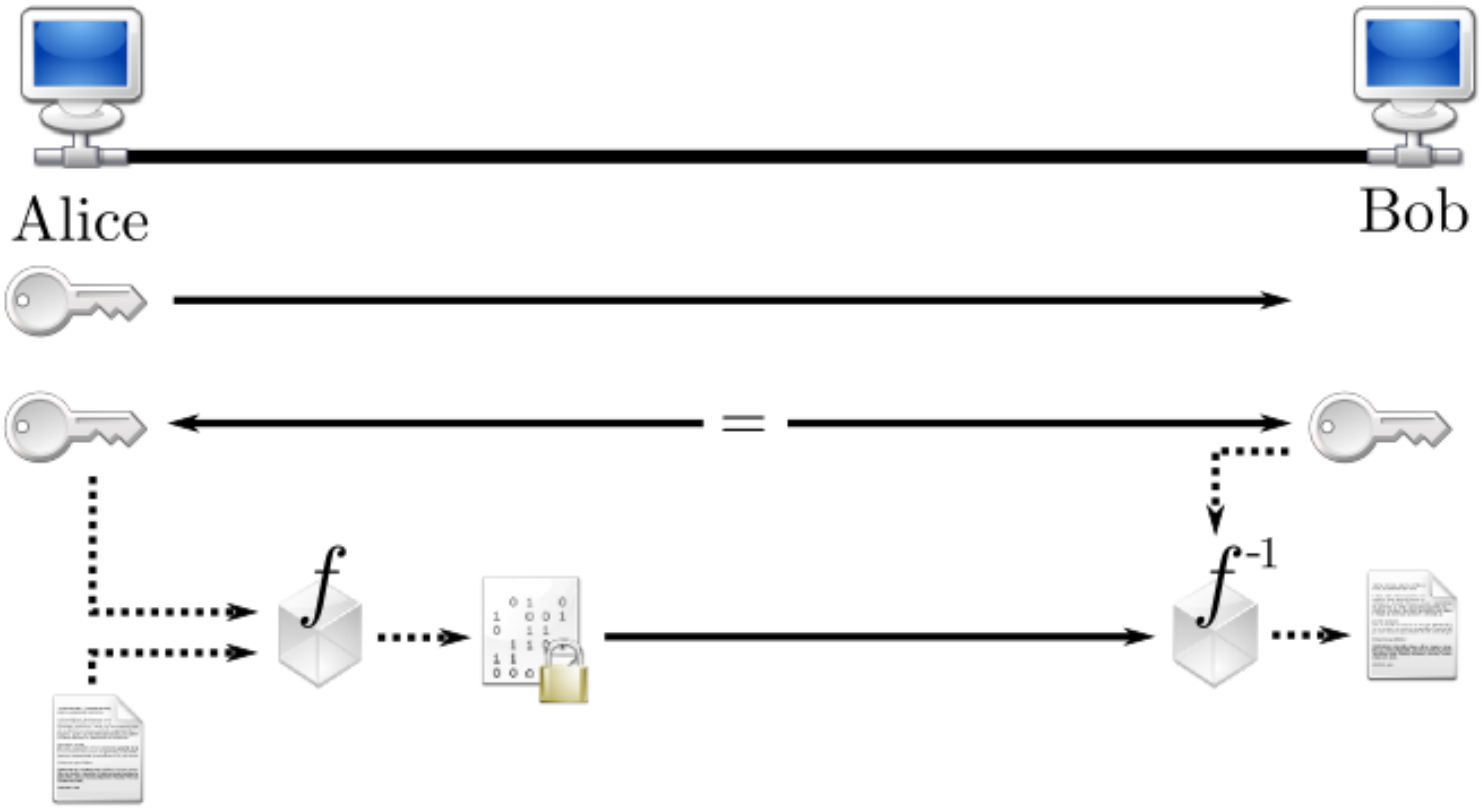
Theoretischer Hintergrund

Ein Streifzug durch die Kryptographie mit Alice, Bob und Eve

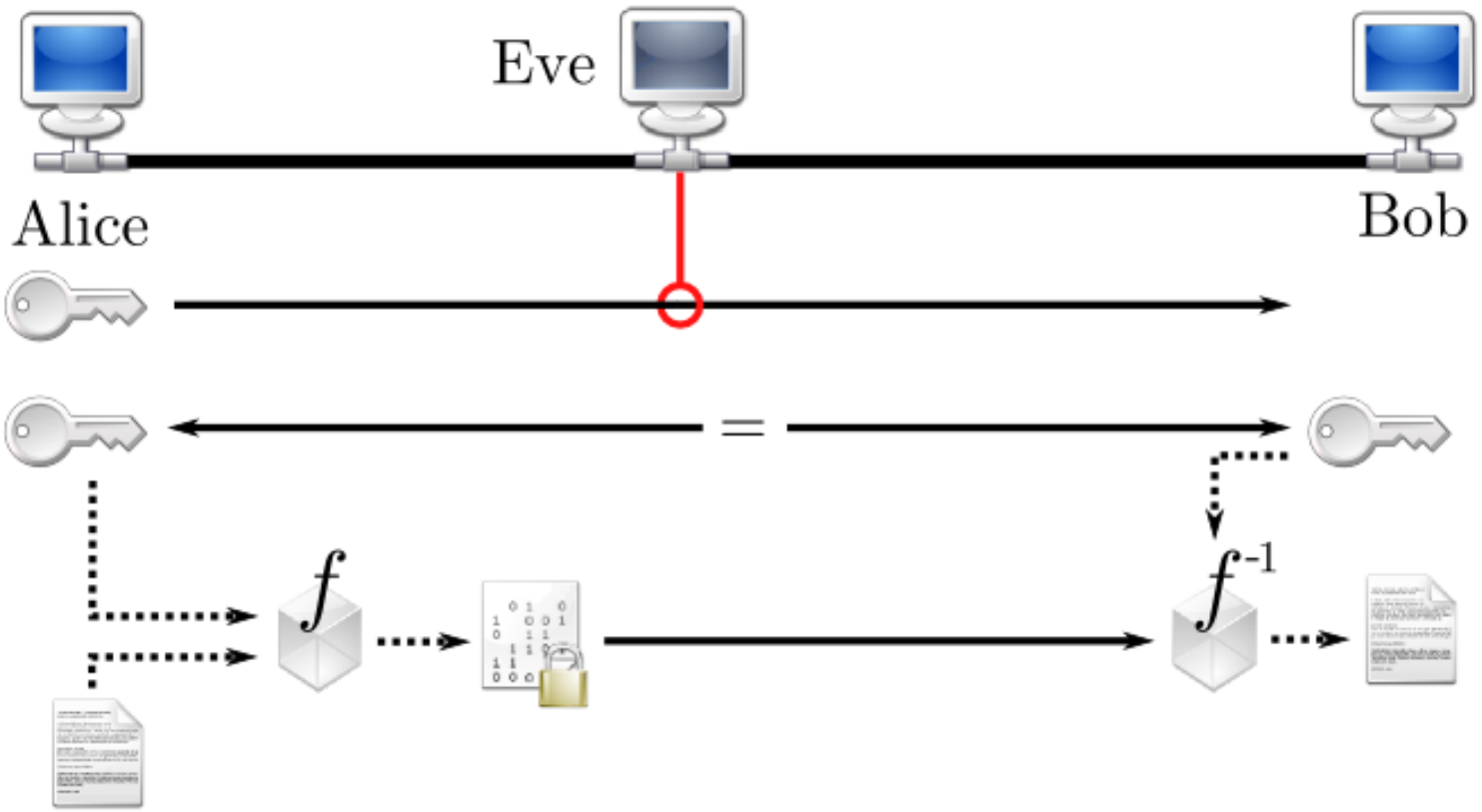
Erwünschte Eigenschaften

- secrecy/confidentiality: nur Berechtigte können Nachrichten entschlüsseln
  - perfect forward secrecy: die Kenntnis des aktuellen Schlüssels ermöglicht nicht, vergangene oder zukünftige Nachrichten zu entschlüsseln
- authentication: Herkunft der Nachricht kann eindeutig verifiziert werden.
  - data integrity: Veränderungen an der Nachricht nach Absenden können bemerkt werden.

Symmetrische Verschlüsselung



Symmetrische Verschlüsselung



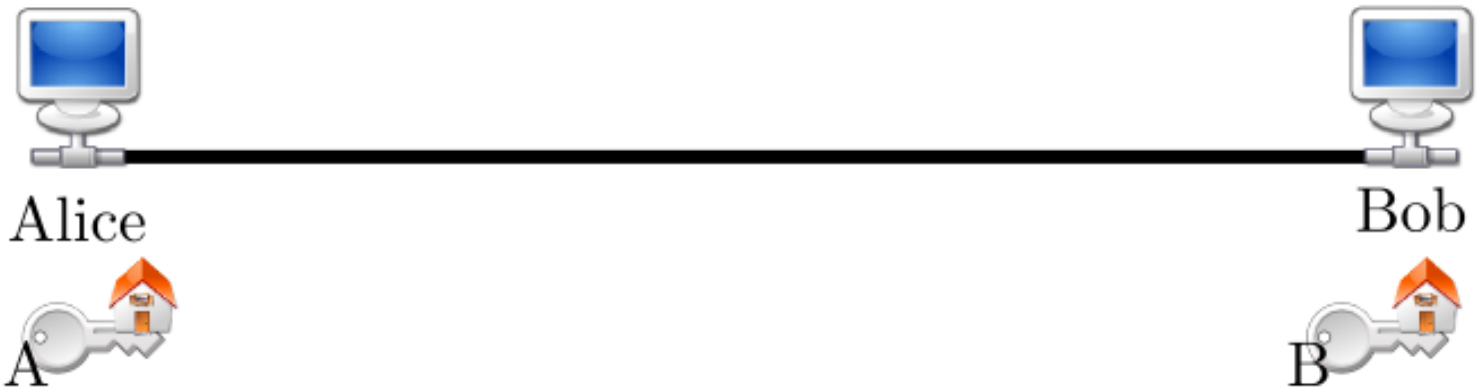
Nachteile der Symm. Verschlüsselung

- Verteilung der Schlüssel muss sicher sein
- Anzahl der Schlüssel steigt ∼quadratisch mit Anzahl der Kommunikationspartner n:

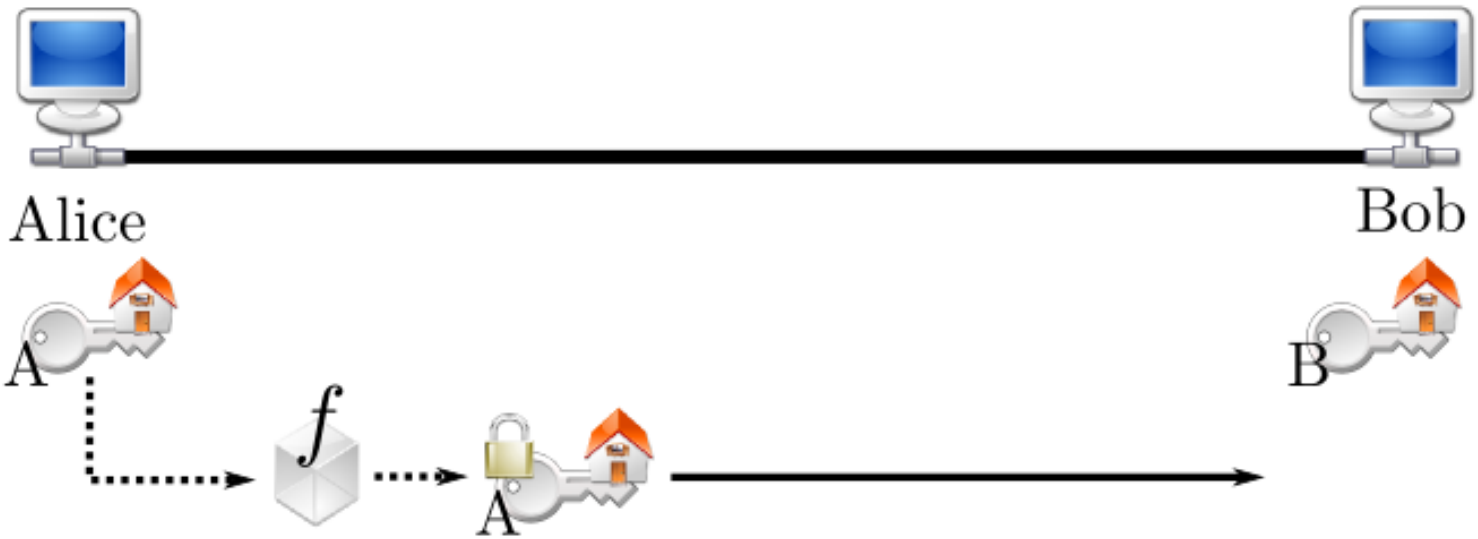
$$|keys| = \frac{n^2 - n}{2}$$

für $n = 100$ ist $|keys| = 4950$

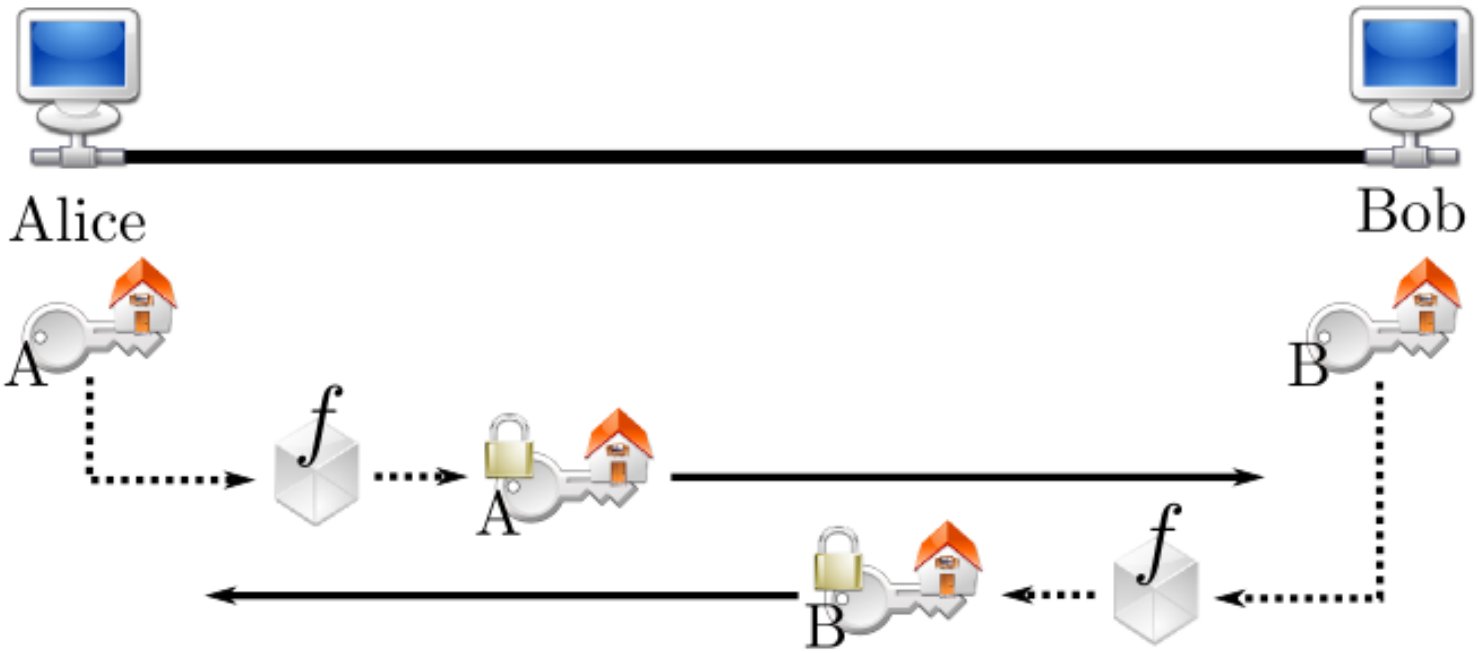
Diffie-Hellman Schlüsselaustausch



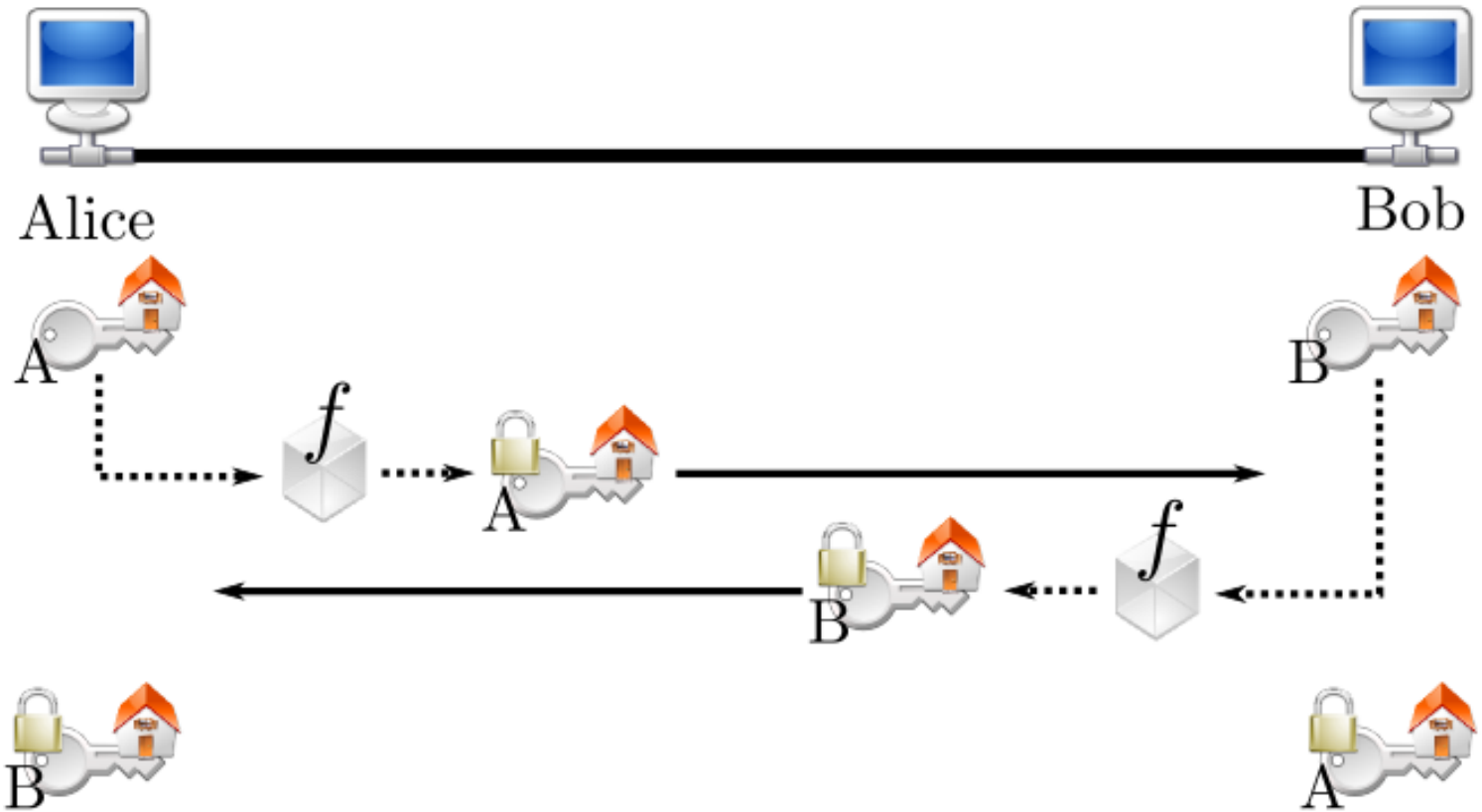
Diffie-Hellman Schlüsselaustausch



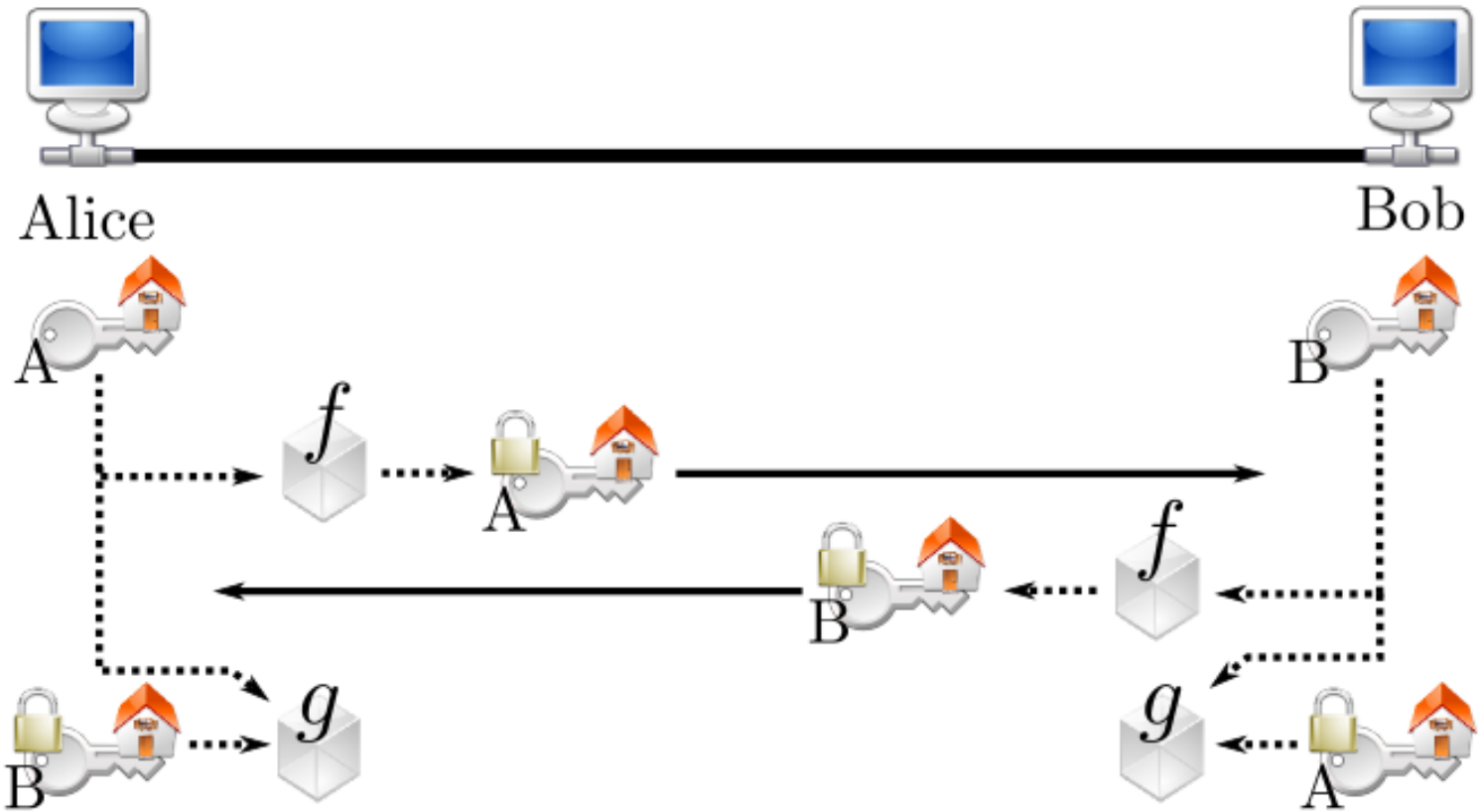
Diffie-Hellman Schlüsselaustausch



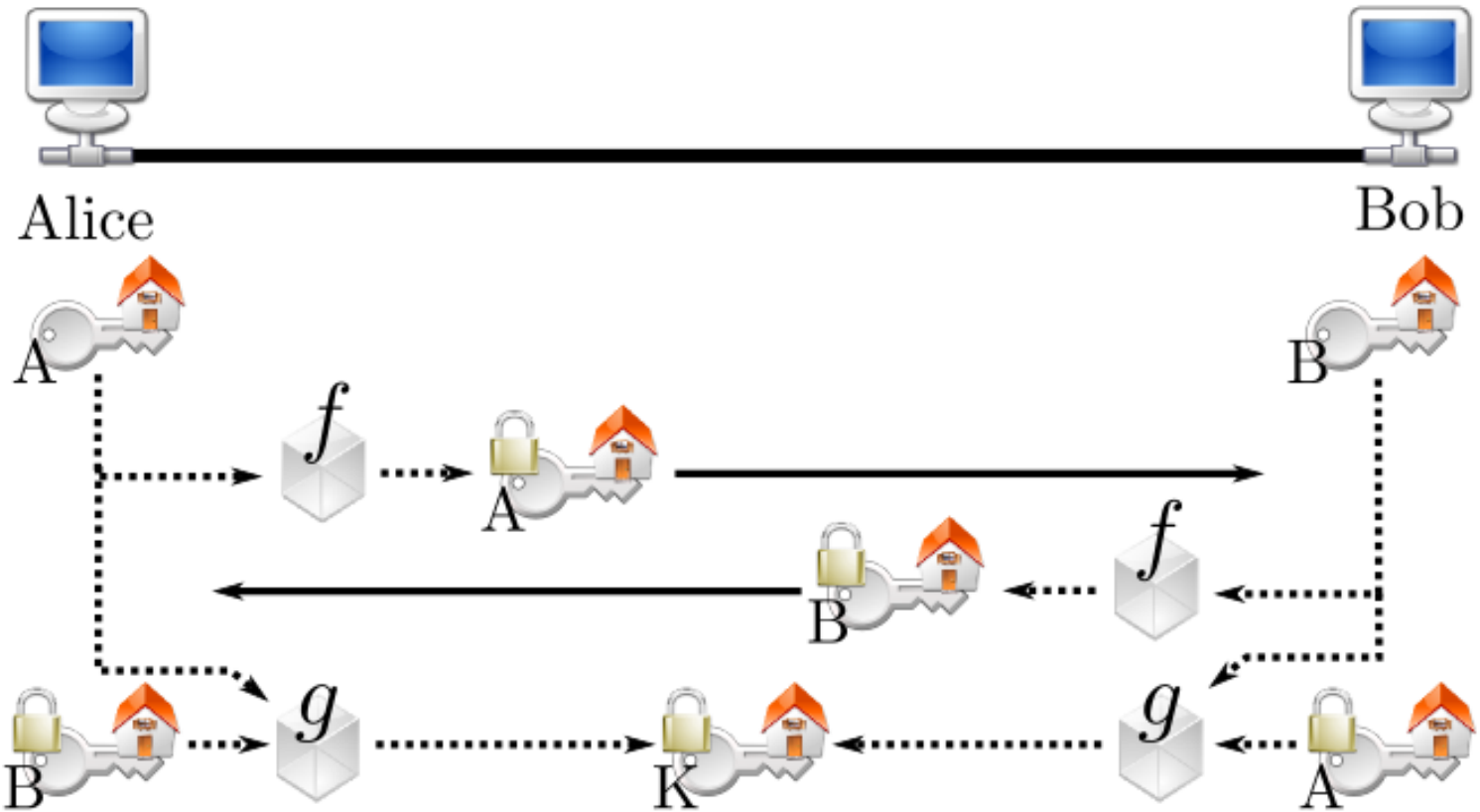
Diffie-Hellman Schlüsselaustausch



Diffie-Hellman Schlüsselaustausch



Diffie-Hellman Schlüsselaustausch



Eigenschaften DH Schlüsselaustausch

- beide Seiten erhalten gleiche Zahl, ohne den Ursprungsschlüssel A bzw. B zu erfahren
- Angreifer kann aus $g(A)$ und $g(B)$ nicht effizient auf A oder B schließen ($\rightarrow$ discrete logarithm problem)

Diffie-Hellman Schlüsselaustausch, Funktionen f, g

- gegeben: Primzahl p
- $\mathbb{Z}_p^*$: Gruppe mit Menge $\mathbb{M} = \{1, 2, \ldots, p-1\}$ und Operation: Multiplikation modulo p
- gegeben: erzeugendes Element e für die Gruppe
- Beispiel: $p = 5, \quad \mathbb{M} = \{1, 2, 3, 4\}, \quad e = 2$
  e ist erzeugendes Element, da:

$$e^0, e^1, e^2, e^3, e^4 = 1, 2, 4, 3, 1$$

Diffie-Hellman Schlüsselaustausch, Funktionen f, g

- für DH sind p und e öffentlich gegeben
- zum Schlüsselaustausch wählt jede Seite eine zufällige Zahl aus $\mathbb{Z}_p^*$ (hier: x und y)
- $f(x) = e^x$ bzw. $f(y) = e^y$ werden an Gegenseite geschickt
- $g(x, e^y) = (e^y)^x = e^{xy}$ und $g(y, e^x) = (e^x)^y = e^{yx}$ werden berechnet

Public Key Kryptographie

Asymmetrische Verschlüsselung:

- privater (secret) Schlüssel zum Entschlüsseln (S)
- öffentlicher (public) Schlüssel zum Verschlüsseln (P)

S und P bestimmen den Ent- und Verschlüsselungsprozess D und E mit den Eigenschaften:

- $D \circ E = id$
- D und E effizient realisierbar
- S lässt sich „nur schwer“ aus P ableiten

Anwendung: Sender berechnet für Nachricht M: $M' = E(M)$, sendet Ergebnis an Empfänger, diese erhält Nachricht aus $D(M') = D(E(M)) = M$.

Signaturverfahren mit öffentl. Schlüssel

- privater Schlüssel (S)
- öffentlicher Schlüssel (P)

Eigenschaften:

- $D \circ E = id$
- D und E effizient realisierbar
- S lässt sich „nur schwer“ aus P ableiten
- $\mathbf{E} \circ \mathbf{D} = \mathbf{id}$

Anwendung: Sender berechnet für Nachricht M Signatur s: $s = D(M)$, sendet Ergebnis an Empfänger, diese überprüft Sinatur mit: $E(s) = E(D(M)) = M$.

Datenintegrität

- Integrität der Daten wird in Transportschicht durch *message authentication code*s (MAC) gewährleistet.
- MAC$\approx$ Hashfunktion mit (geheimen) Schlüssel $\rightarrow$ MAC zu Nachricht m kann nur mit Kenntnis des Schlüssels berechnet werden
- ideale MAC-Funktion ordnet Eingabedaten zufällige Ausgabe zu.
- in SSH2 wird folgendes berechnet: `mac = MAC(key, sequence_number ||unencrypted_packet)`
- SSH1 benutzte CRC32 als Überprüfung der Integrität $\rightarrow$ compensation attack möglich

Funktionsweise von SSH

Umsetzung der spezifizierten Eigenschaften

Ablauf einer ssh-session, ssh-transport

1. client verbindet zu server port 22 (default)
2. client und server tauschen Identifikationsstring aus (SSH-protocolVersion-softwareVersion-comments) (SSH-1.99-OpenSSH_2.9p1)
3. key exchange wird durchgeführt
   $\Rightarrow$ shared secret: K und Hash: H $\Rightarrow$ Verschlüsselungs- und Authentifizierungsschlüssel; H dient zudem als session identifier
4. server schickt Signatur von H mit private-hostkey an client
5. client schickt service request an server (z.B. „ssh-userauth“)

Ablauf einer ssh-session, ssh-userauth

1-5 Transport layer

6. server sendet Liste von verfügbaren Authentifizierungsmethoden
7. client versucht Authentifizierung zu erfüllen z.B. durch senden einer Signatur erstellt mit einem private key
8. Wdh. Schritt 7, bis Server genügend gültige Authentifizierungen erhalten hat (signalisiert mit SSH-MSG-USERAUTH-FAILURE und Liste der noch ausstehenden Schritte bzw. SSH-MSG-USERAUTH-SUCCESS)

Ablauf einer ssh-session, ssh-connect

1-5 Transport layer

6-8 User authentication layer

9. client oder server öffnen channel (SSH_MSG_CHANNEL_OPEN)
10. nach positiver Bestätigung der Gegenseite können Nutzdaten gesendet werden (SSH_MSG_CHANNEL_DATA)
11. Ende der Datenübertragung und Ende der Verbindung wird mit SSH_MSG_CHANNEL_EOF- und SSH_MSG_CHANNEL_CLOSE-Paketen signalisiert

Host authentication

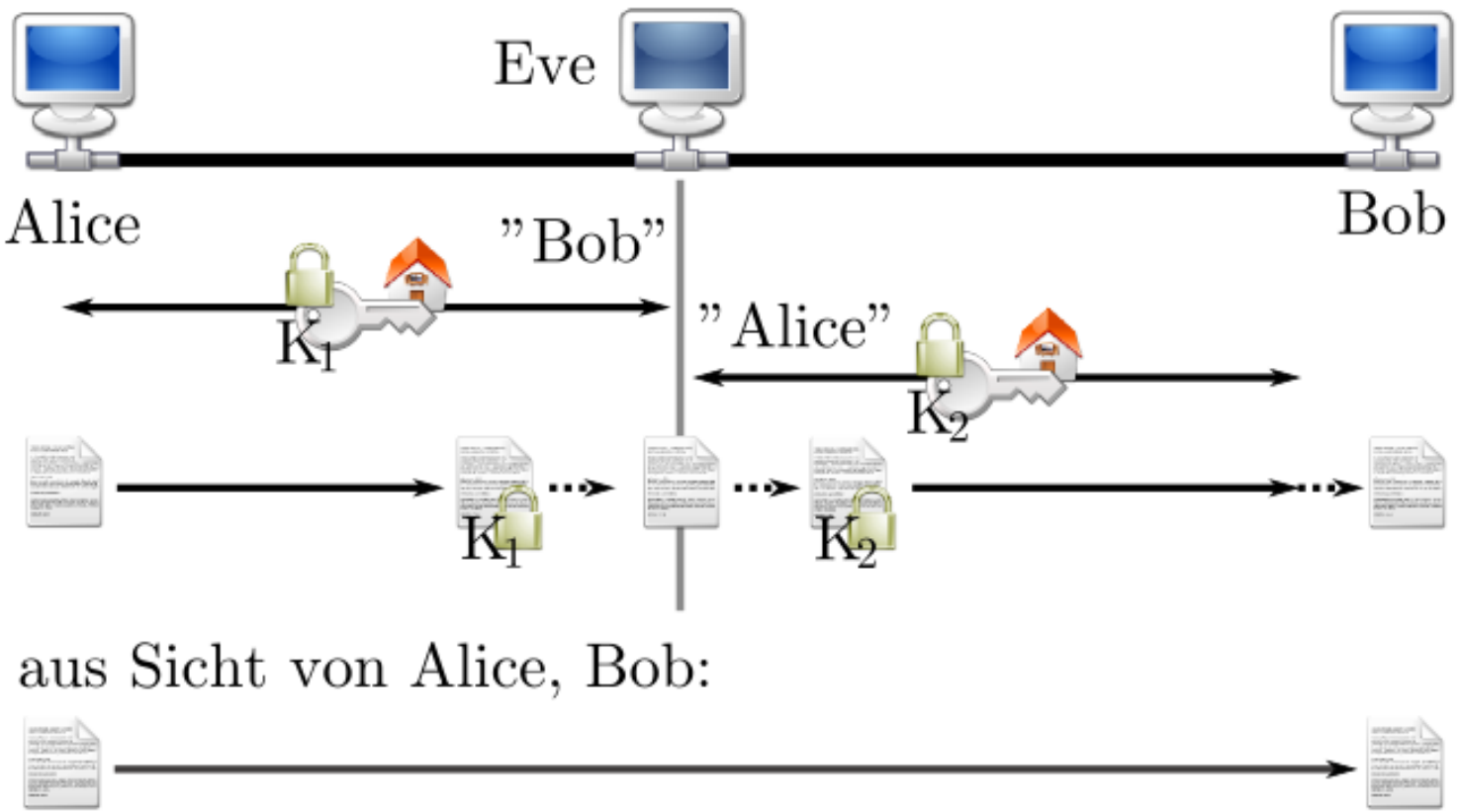
- Server authentifiziert sich bei client mit der Signatur seines public (host) keys
- Signatur muss bei erster Verbindung bestätigt werden und wird in ~/.ssh/known_hosts (default) gespeichert
- ändert sich Signatur eines Servers bei späterer Verbindung, wird eine Warnung ausgegeben
- Überprüfung der Signatur relevant, um man in the middle Angriffe zu vermeiden

Host authentication, Beispiel

```
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.
```

Man in the Middle Attack



User Authentication

- Passwort
- Public Key zusammen mit Passphrase (Benutzerbasiert)
- Hostbasiert

Anwendung

SSH in der Praxis

ssh

Remote login:

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

... mit explizitem Benutzernamen:

```
bob:/$  ssh alice@linux.uni-koblenz.de
```

ssh

Remote login:

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

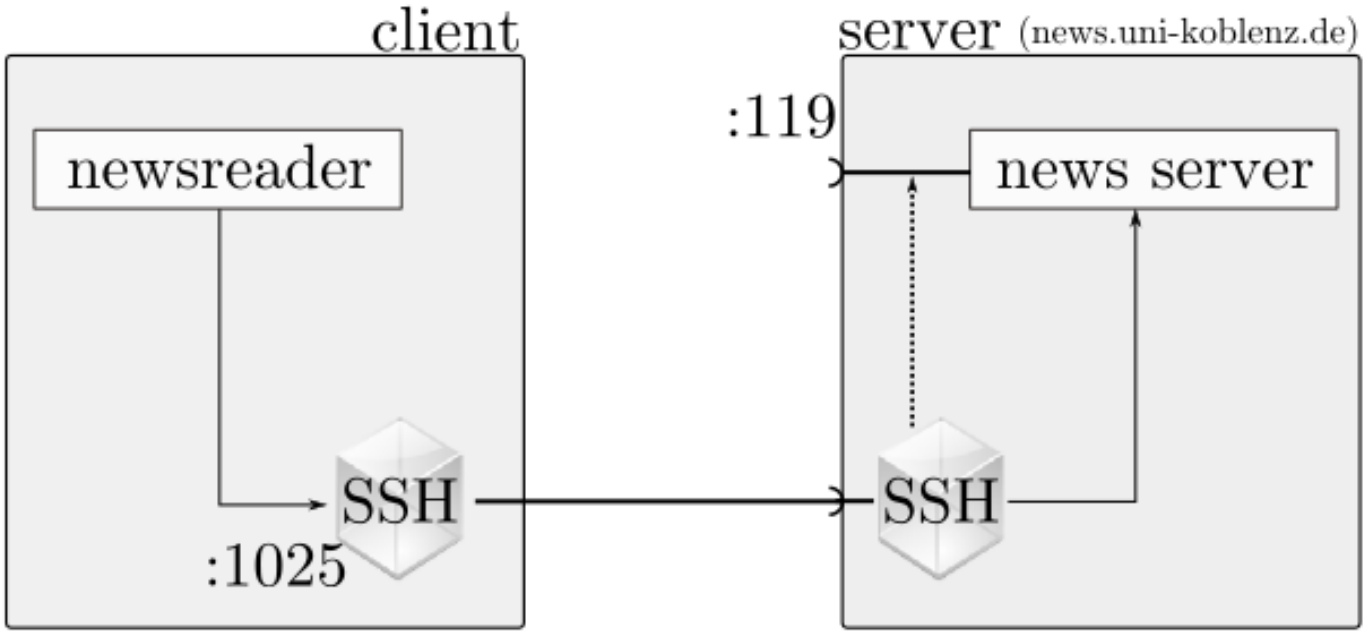
... mit explizitem Benutzernamen:

```
bob:/$  ssh alice@linux.uni-koblenz.de
```

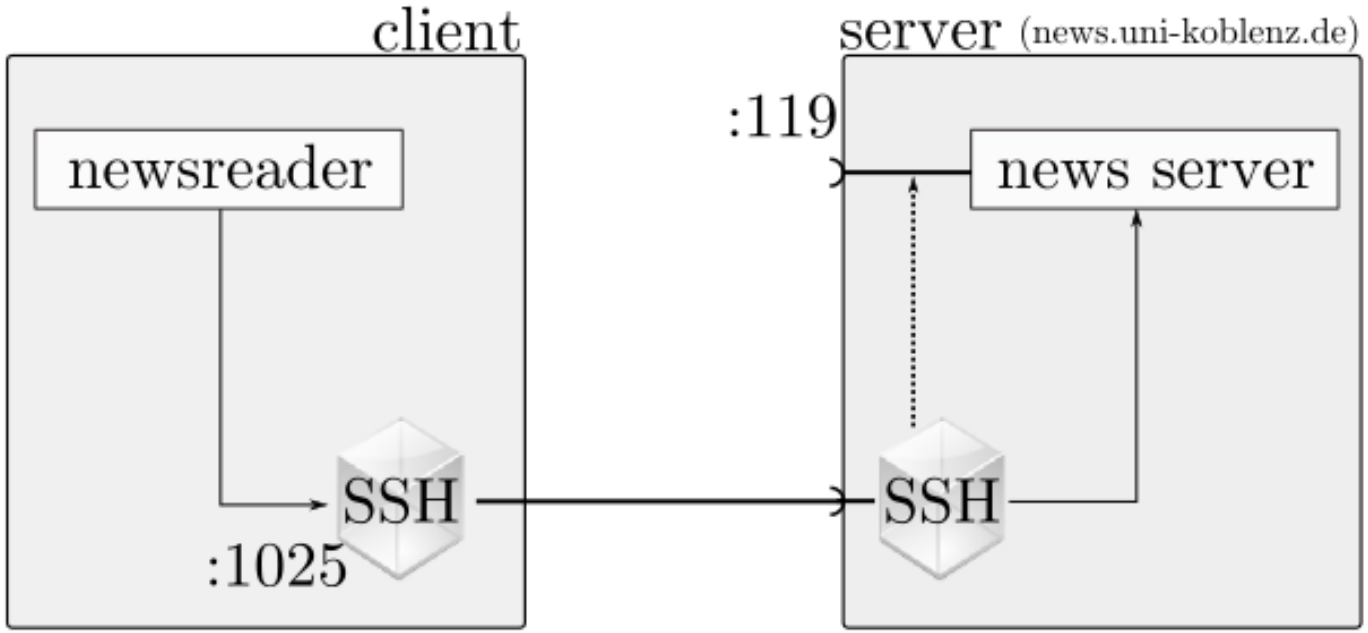
Programmausführung auf Host:

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

Port Forwarding, Beispiel



Port Forwarding, Beispiel



Port-forwarding (Newsserver auf lokalen Port 1025):

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

Konfiguration des ssh-servers

Beispielinhalt der Konfigurationsdatei:
(Default: /etc/ssh/sshd_config)

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

scp

`scp` kopiert nicht-interaktiv Dateien zwischen zwei Hosts.
Syntax:

```
scp source destination
```

konkret:

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

Benutzerauthentifizierung mit Public Keys

- Benutzer generiert mit `ssh-keygen` ein PK-Schlüsselpaar:

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

- auf dem Remote Account muss der public key der Datei `~/.ssh/authorized_keys` hinzugefügt werden
- bei login auf diesem Account wird dann die passphrase des PK-Schlüsselpaars abgefragt, um Signatur zu erstellen

Demo

$\longrightarrow$X11 forwarding, port forwarding

Literatur


Daniel J. Barrett and Richard E. Silverman.
SSH: The Secure Shell: The Definitive Guide.
2001.


Michael D. Bauer.
Building Secure Servers with Linux.
2002.


Bruce Schneier.
Applied Cryptography: Protocols, Algorithms, and Source Code in C.
John Wiley & Sons, Inc., New York, NY, USA, 1993.


Niels Ferguson and Bruce Schneier.
Practical cryptography.
John Wiley & Sons, Inc., 2003.

[2] [1] [4] [3]