SSH-Verbindungen

Secure Shell oder SSH bezeichnet ein kryptographisches Netzwerkprotokoll für den sicheren Betrieb von Netzwerkdiensten über ungesicherte Netzwerke.

Hinweise: https://wiki.ubuntuusers.de/SSH/

Empfehlenswerte Einstellungen für einen Server, der über das Internet erreichbar ist (z.B. ein Webserver):

- Den direkten Zugang für „root“ von Außen ausschließen.
- Den Zugang mit Passwort von Außen generell ausschließen.
- Zugang nur mit Schlüssel.
- Optional z.B. für Webserver: Beschränkter Zugriff über SFTP auf definierte Bereiche (HTML-Files …)

SSH-Schlüssel

Für den Zugriff mit einem Schlüssel muss zunächst einer generiert werden. Dabei wird i.d.R. ein Schlüsselpaar generiert, das aus einem privaten und einem öffentlichen Schlüssel besteht und das miteinander agiert. Der private Schlüssel bleibt lokal und geheim, der öffentliche wird an externe Systeme verteilt.

SSH-Key unter Linux generieren

Auf einem (lokalen) Linux-System das Programm open-ssh-client installieren und Schlüssel generieren.

```
sudo apt install openssh-client
```

```
ssh-keygen -t rsa -b 4096
```

t = Typ (hier rsa)
b = Schlüssellänge (hier 4096 Bit)

Sofern nicht anders angegeben, liegt der neue Schlüssel automatisch im (versteckten) Verzeichnis ~/.ssh/
Den Schlüssel ohne Passwort zu generieren, vereinfacht das spätere Login, da dann kein Passwort mehr angegeben werden muss. Die 2FA wird aber empfohlen.
Soll der private Schlüssel auf ein anderes Systeme kopiert werden, um ihn auch von dort nutzen zu können: Auf dem neuen System müssen die Zugriffsrechte genauso eingeschränkt werden. Anderfalls gibt es Verbindungsprobleme.
privater Schlüssel: .ssh/id_rsa -rw——- Owner & Group = User

öffentl. Schlüssel: .ssh/id_rsa.pub -rw-r–r– Owner & Group = User

Der private Schlüssel muss unbedingt vor fremdem Zugriff geschützt bleiben!
Das betrifft auch den Transfer der Dateien (USB-Stick, E.Mail, etc.)
Versand per E-Mail nur mit verschlüsselter E-Mail!

Bei der Erstellung werden 2 Dateien angelegt:

- id_rsa (privater Schlüssel)
- id_rsa.pub (öffentlicher Schlüssel)

Der öffentliche Schlüssel wird auf das entfernte System übertragen, auf das zugegriffen werden soll. Der private Schlüssel bleibt auf dem lokal System, auf dem er generiert wurde. Für jedes weitere (lokale) System sollte jeweils ein eigener Schlüssel generiert werden.
Werden privater und öffentllicher Schlüssel auf einen anderen PC kopiert, kann auch von dort aus auf die Server zugegriffen werden - ohne das der öffentliche Schlüssel neu auf diese Server übertragen werden muss.

Zur Übertragung auf einen Server muss der User bereits dort angelegt sein und der Zugriff ohne Schlüssel (mit Passwort) muss temporär freigegeben werden.
Den öffentlichen Schlüssel (id_rsa.pub) wie folgt auf den Server übertragen:

```
ssh-copy-id <USER>@<REMOTEHOST>
```

Ersetzen: <USER> und <REMOTEHOST>
Das Passwort vom <REMOTEHOST> wird abgefragt.

Im <REMOTEHOST>-Home-Verzeichnis vom <USER> liegt die Datei **~/.ssh/authorized_keys**. In diese Datei werden die gültigen Public-Keys (automatisch) eingetragen. Das Verzeichnis ist versteckt.
Parallel wird auf dem lokalen (Linux-)Rechner der (neue) entfernte Host in der Datei **~/.ssh/known_hosts** aufgenommen. Ist der Host dort bereits enthalten, ggf. mit anderem Schlüssel, muss er zunächst aus dieser Datei entfertn werden → Siehe Fehlermeldung und Hinweise.

SSH-Key mit Putty generieren

Alternativ ist es möglich, mit dem Programm PuTTYgen, z.B. unter Windows, einen Schlüssel zu erstellen. Es ist möglich, den angezeigten Block direkt aus PuTTYgen heraus in die entfernte **~/.ssh/authorized_keys** des Servers zu kopieren - ggf. die Datei neu erstellen.

Die Datei hat folgende Struktur (alles hintereinander):

- ssh-rsa « dieser Text und 1 Leerzeichen
- rsa-pub-key « der eigentliche Schlüssel aus Puttygen
- Key Kommentar « Im Textblock von puttygen bereits enthalten

Liegt bereits ein SSH-Key vor (z.B. erstellt wie oben beschrieben), kann dieser auch für den Zugriff mit Putty oder SFTP umgewandelt werden.
Programm PuTTYgen: Private-key importieren und als PuTTY-private-key speichern.

Schlüssel von PuTTY können von diversen Systemen (FileZilla, etc.) genutzt werden, sofern der Public-Key im entfernten Server hinterlegt ist. Da dieser Schlüssel kopier- und übertragbar ist, sollte er immer zusätzlich mit einem Passwort geschützt sein.

Für Konvertierung Programm **PuTTYgen** aufrufen.

1. Load an existing private key file
2. Save private key » jetzt mit Endung .ppk

Server konfigurieren

SSH-Zugriffe

Ggf. vorher Installieren

```
sudo apt-get install openssh-server
```

```
sudo nano /etc/ssh/sshd_config
```

```
ClientAliveInterval 1200
ClientAliveCountMax 3

PermitRootLogin no
PasswordAuthentication no
Subsystem sftp internal-sftp
```

Um sich nicht selber auszusperren:
PermitRootLogin nur deaktivieren, sofern ein anderer User Zugriff hat und PasswordAuthentication nur abschalten, sofern der Zugriff mit dem Key-File auch klappt!

Möglicherweise gibt es Parameter in einem Unterordner, die die Regeln überschreiben. Möglich, dass dort in einer *.conf-Datei,
z.B. /etc/ssh/sshd_config.d/50-cloud-init.conf, hinterlegt ist: **PasswordAuthentication yes**
Das muss dann angepasst werden.

The **ClientAliveInterval** parameter specifies the time in seconds that the server will wait before sending a null packet to the client system to keep the connection alive.
The **ClientAliveCountMax** parameter defines the number of client alive messages which are sent without getting any messages from the client.
Timeout value = ClientAliveInterval * ClientAliveCountMax
Beispiel: 1200 x 3 = 3600 ~ 1 Stunde.
Nach Änderungen muss der SSH-Service neu gestartet werden.

```
sudo systemctl reload ssh
```

SSH-Zugriff auf VPN beschränken

> **Debian 13 Server (nur IPv6 öffentlich)**
> **SSH (und SFTP) ausschließlich über WireGuard-VPN (IPv4 10.8.0.0/24)**
> **Firewall: UFW**

🎯 Zielbild (Soll-Zustand)

- SSH **nur erreichbar über WireGuard**
- VPN-Netz: `10.8.0.0/24`
- **Kein** öffentlicher SSH (weder IPv4 noch IPv6)
- UFW aktiv
- SSH zusätzlich **hart an VPN gebunden**

🧱 Voraussetzungen

- Du hast **Konsolenzugriff** oder funktionierenden VPN-Zugang
- WireGuard-Interface heißt `wg0`
- Server-VPN-IP z. B. `10.8.0.1`

🔍 Schritt 1 – WireGuard prüfen

```
ip a show wg0
```

Erwartet:

```
inet 10.8.0.1/24 scope global wg0
```

🔥 Schritt 2 – UFW installieren & Grundregeln

```
sudo apt update
```

```
sudo apt install ufw
```

Default-Policies:

```
sudo ufw default deny incoming
```

```
sudo ufw default allow outgoing
```

🔓 Schritt 3 – WireGuard selbst freigeben

(typisch: UDP 51820 – ggf. anpassen)

```
sudo ufw allow 51820/udp
```

✅ Schritt 4 – SSH **nur** über WireGuard erlauben

(empfohlen: Interface-gebunden)

```
sudo ufw allow in on wg0 to any port 22 proto tcp
```

Alternativ:

```
sudo ufw allow from 10.8.0.0/24 to any port 22 proto tcp
```

✅ Schritt 5 – Öffentlichen SSH entfernen (IPv4 + IPv6)

Status anzeigen:

```
sudo ufw status numbered
```

Du wirst sehen:

```
22/tcp (OpenSSH)       ALLOW IN Anywhere
22/tcp (OpenSSH (v6)) ALLOW IN Anywhere (v6)
```

Löschen:

```
sudo ufw delete allow ssh
```

oder gezielt per Nummer:

```
sudo ufw delete <NUMMER>
```

✅ Schritt 6 – IPv6 in UFW deaktivieren (wichtig!)

Da SSH **nur über IPv4-VPN** laufen soll:

```
sudo nano /etc/default/ufw
```

Ändern:

```
IPV6=no
```

Dann:

```
sudo ufw reload
```

✅ Schritt 7 – UFW aktivieren

```
sudo ufw enable
```

Prüfen:

```
sudo ufw status verbose
```

Soll:

```
22/tcp     ALLOW IN    on wg0
51820/udp  ALLOW IN    Anywhere
```

Schritt 8 – SSH **hart an WireGuard binden**

Das ist der entscheidende Sicherheitsanker.

```
sudo nano /etc/ssh/sshd_config
```

Eintragen:

```
AddressFamily inet
ListenAddress 10.8.0.1
```

Dann:

```
sudo systemctl restart ssh
```

Schritt 9 – Tests (Pflicht!)

Ohne VPN

```
ssh user@SERVER_IPV6
```

➡ muss fehlschlagen

Mit VPN

```
ssh user@10.8.0.1
```

➡ muss funktionieren

Schritt 10 – Finale Kontrolle

```
sudo ss -tlnp | grep :22
```

Soll:

```
LISTEN 10.8.0.1:22
```

 nicht erlaubt:

```
0.0.0.0:22
[::]:22
```

Warum dieses Setup Best Practice ist

Ebene	Schutz
WireGuard	Zugang nur für autorisierte Clients
UFW	Filtert Traffic
SSH `ListenAddress`	verhindert offenen Port technisch
IPv6 deaktiviert	keine „stille“ Umgehung

➡ Selbst bei Firewall-Fehlern kein öffentlicher SSH möglich

Kurzfassung (Merkliste)

1. VPN zuerst testen
2. SSH nur wg0 erlauben
3. OpenSSH (v6) löschen
4. IPv6 in UFW deaktivieren
5. SSH an 10.8.0.1 binden

Schutz vor Angriffen mit fail2ban

SSH Login schützen mit fail2ban
How To Protect SSH with Fail2Ban on Debian 11
Installation und Verwendung von Fail2ban unter Debian 12
fail2ban bei Ubuntu-Users

fail2ban installieren

```
sudo apt update
```

```
sudo apt install fail2ban
```

Conf-Dateien kopieren

```
sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
```

```
sudo cp /etc/fail2ban/fail2ban.conf /etc/fail2ban/fail2ban.local
```

nur die .local-Dateien bearbeiten

```
sudo nano /etc/fail2ban/jail.local
```

Ändern (nach [sshd]) suchen:

[sshd]

```
backend=systemd
enabled = true
port = ssh
filter = sshd
logpath = /var/log/auth.log
maxretry = 3
```

Über die Zeit-Parameter in der Datei /etc/fail2ban/jail.local lässt es sich steuern:
„bantime“ is the number of seconds that a host is banned.
Beispiel: Sperrung(banned) für 1 Stunde, wenn maxrtry innerhalb findtime erreicht wurde
bantime = 3600
A host is banned if it has generated „maxretry“ during the last „findtime“
Beispiel: Zeit (hier 3 Minuten).
findtime = 180
„maxretry“ is the number of failures before a host get banned.
maxretry = 5

Installieren

```
sudo apt install python3-systemd
```

fail2ban neu starten

```
sudo systemctl restart fail2ban
```

Autostart mit System

```
sudo systemctl enable fail2ban
```

Protokoll der (temporär) verbanten IP-Adressen

```
sudo zgrep 'Ban' /var/log/fail2ban.log*
```

Verbindungs-Protokolle

Quelle: https://www.strongdm.com/blog/view-ssh-logs

If you want to view ssh logs from a specific time range, you can use the since and until flags. Some examples:

```
sudo journalctl -u ssh --since yesterday
```

```
sudo journalctl -u ssh --since -3d --until -2d # logs from three days ago
```

```
sudo journalctl -u ssh --since -1h # logs from the last hour
```

```
sudo journalctl -u ssh --until "2024-12-20 07:00:00"
```

To watch the ssh logs in realtime, use the follow flag:

```
sudo journalctl -fu ssh
```

Use Ctrl-C to exit out of the log monitor.

Login

Login über Linux-Shell

Login mit Passwort:

```
ssh <USER>@<REMOTEHOST>
```

<USER>: Benutzername auf Remote-Host. <USER>@ kann weggelassen werden, wenn entfernter <USER> mit dem lokalen Usernamen übereinstimmt.
<REMOTEHOST>: IP-Adresse des Remote-Host

Login mit Key wenn der key im <REMOTEHOST> bereits hinterlegt ist:

```
ssh -i <KEY_PFAD> <USER>@<REMOTEHOST>
```

<KEY_PFAD> z.B.: .ssh/id_rsa
Standardpfad für den Key ist: .ssh/id_rsa wenn er dort liegt, kann „-i <KEY_PFAD>“ weggelassen werden.
i = identity_file

Beim ersten Login, wenn der public-key noch nicht auf dem Server ist oder dieser sich geändert hat, muss dieser im Remote-Server registriert werden.

```
ssh-copy-id <USER>@<REMOTEHOST>
```

Beim ersten Login erfolgt eine Validierung mit dem Passwort des Systems. Bei Folgeaufrufen nur noch mit dem PW des SSH-Keys bzw. wenn keines vergeben wurde, ohne PW.

Beim ersten Login werden die dann bekannten Hosts lokal in **~/.ssh/known_hosts** gespeichert. Gibt es Änderungen an einem Host und ggf. damit verbundene Probleme, dann kann der Host daraus, oder die ganze Datei, gelöscht werden. Wird dann beim nächsten Aufruf neu generiert. Auf dem server wird der „neue“ Key eines Users eintgetragen in der Datei .ssh/authorized_keys. Für jeden User des Server-Systems werden die Schlüssel separat in seinem Home-Verzeichnis verwaltet.

Wurde der Schlüssel am Server geändert, oder der Server neu eingerichtet, muss er aus der lokalen Datei ~/.ssh/known_hosts ausgetragen werden. Händisch oder mit dem Befehl (IP des betroffenen Servers):

```
ssh-keygen -f "~/<USER>/.ssh/known_hosts" -R "<REMOTEHOST>"
```

Login mit PuTTY

Mit PuTTY die Verbindung wie folgt definieren:

- <REMOTEHOST>
- Port (weglassen, wenn 22 - Normalfall)
- SSH
- Name (Saved Session)
- /Connection/SSH/Auth/ » Laden: Private Key File[1)]
- Option: /Connection/Data/ » Auto-Login username = <USER>
- Option: /Connection/ » Secons between keepalives = 600 (verhindert das auto-lockout)
- Zurück auf „Session“ und **Save**

Login mit FileZilla

Für den Zugriff kann die mit Puttygen generierte .ppk-Datei genutzt werden.
Verbindungsart: **Schlüsseldatei**.

Dateien kopieren über SSH

Dafür nicht vorab auf dem Remote-Server einloggen, sondern vom lokalen Rechner ausführen.

Kopieren der Datei “foobar.txt” von einem entfernten Rechner auf den lokalen Rechner.

```
scp <USER>@<REMOTEHOST>:foobar.txt /some/local/directory
```

Kopieren der Datei “foobar.txt” vom lokalen Rechner auf einen entfernten Rechner.

```
scp foobar.txt <USER>@<REMOTEHOST>:/some/remote/directory
```

Kopieren der Datei “foobar.txt” vom Remote-Host “<REMOTEHOST_1>„ auf den Remote-Host “<REMOTEHOST_2>”.

```
scp <USER>@<REMOTEHOST_1>.edu:/some/remote/directory/foobar.txt \
<USER>@<REMOTEHOST_2>:/some/remote/directory/
```

Einzelne Verzeichnisse kopieren.
Kopieren des Verzeichnisses “foo” vom lokalen Rechner in das Verzeichnis “bar” eines entfernten Rechners.

```
scp -r foo <USER>@<REMOTEHOST>:/some/remote/directory/bar
```

Quelle: https://www.davidkehr.com/linux-kopieren-von-und-zu-einem-computer-per-scp-ssh/

1)
Mit Puttygen generierte .ppk-Datei