



DataBASHing - Remotezugriff auf externe Rechner/Dateien

Martin Raden

Einführung von

- `wget`, `curl` in Zugriff auf Webressourcen
- `ssh` in SSH Zugriff
- `scp`, `rsync` in Dateitransfer im Netz

[Video: \(en\) Corey Schafer - How To Use The rsync Command - Sync Files Locally and Remotely \[18 min\]](#)

Zugriff auf Webressourcen

Dateidownload

`wget` - expliziter Dateidownload ohne Logindaten

- “`-O`” (capital O!) = output = Name des Ausgabedatei (ansonsten Dateiname aus Link oder komplette URL als Dateiname)
- “`-i`” = input = alternative: Name einer Datei, die mehreren URLs von Zieldateien zum Download listet (dann keine URL in Aufruf angegeben)
- “`-c`” = continue = Download weiterführen (wenn zB. unterbrochen durch Netzwerkfehler, reboot, etc.)
- z.B. `wget https://avatars.githubusercontent.com/u/72390537 -O uni-palme.jpg`

Formulare abfragen

`curl` - Abfrage von Formularergebnissen (Eingabemasken)

SSH Zugriff

Verbindungsaufbau

- basiert auf **public-key Verschlüsselung**
 - dazu **zwei Schlüssel** nötig:
 - * mit **public key** werden Daten **verschlüsselt**
 - * mit **private key** werden verschlüsselte Daten **entschlüsselt**

- Verbindungsaufbau immer **zweischrittig**:
 - **(Schritt 1) Verschlüsselung herstellen** (analog zu HTTPS Verbindungsaufbau, s.o.)
 - * Server schickt dazu seinen public key an Client
 - **(Schritt 2) Nutzer-Authentifizierung**
 - * Client schickt dazu seinen public key an Server

Einstiegsartikel: Wie funktioniert SSH?

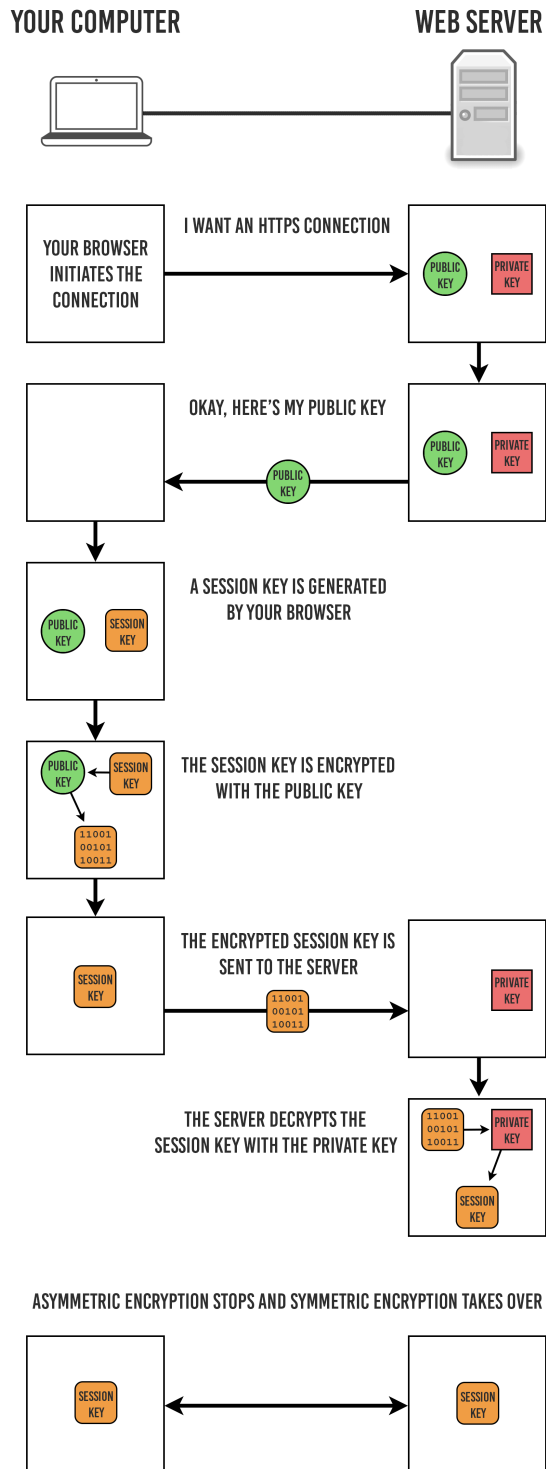
- (www) [Was sind **SSH Schlüssel** und wozu brauche ich sie?](#)
- (www-en) [Was passiert beim Einloggen mittels **SSH Authentifizierung**?](#)
- (www) [HTTPS **Verschlüsselungsaufbau** = Schritt 1](#)
- (www) [Wozu braucht man SSH Verbindungen?](#)

Nachdem sie obige Artikel studiert haben, sollten sie den folgenden beiden zusammenfassenden Infografiken folgen können.

1 - Verschlüsselung

Schritt 1 - Herstellung einer verschlüsselten Verbindung (analog zu HTTPS Verbindungsaufbau)

HOW HTTPS ENCRYPTION WORKS

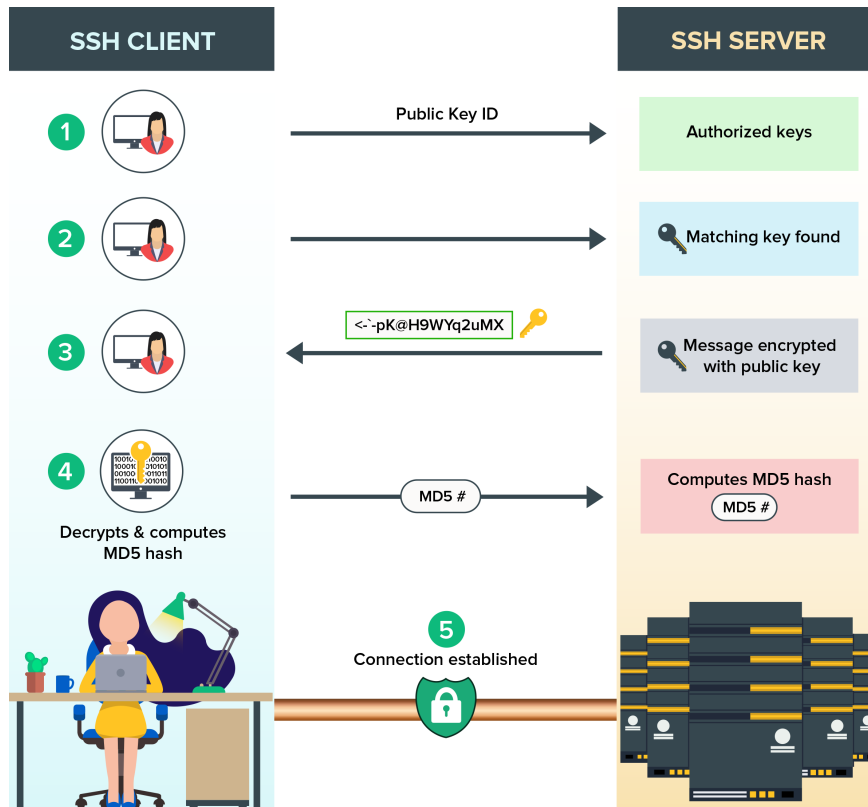


(Grafik von [tiptopsecurity.com](https://www.tiptopsecurity.com))

2 - Authentifizierung

Schritte einer Public Key Authentifizierung

1. [einmalig] public key wird in "authorized_keys" im User account auf Server hinterlegt
2. Verbindung wird initialisiert, dazu Nutzernamen und public key übermittelt. Server prüft, ob public key im "authorized_keys" file des Nutzers vorhanden.
3. Wenn ja, Server verschlüsselt zufällige Nachricht mit dem public key und schickt diese an den User.
4. User entschlüsselt Nachricht mit private key (nur damit möglich) und schickt diese (bzw. dessen MD5 sum) wieder an den Server.
5. Wenn die originale Nachricht und die zurückgeschickte übereinstimmen, ist die Authentifizierung abgeschlossen und der Nutzer erhält Zugriff.



(Grafik von manageengine.com)

SSH Keys

- `ssh-keygen` = **generiert Schlüsselpaar** (private + public) und legt sie im `$HOME/.ssh` Verzeichnis ab
 - Fragen nach Schlüsselname und PassPhrase ggf. einfach mit Eingabetaste (keine Eingabe) bestätigen und (erstmal) Standardwert verwenden
- **Public key** (z.B. in `$HOME/.ssh/id_rsa.pub`) in der Datei `$HOME/.ssh/authorized_keys` **als neue Zeile** eintragen
 - z.B. via `cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys` (damit wird Zielf-datei auch angelegt, wenn diese noch nicht vorhanden ist)

Lokaler SSH Server

- in Ubuntu:
 - prüfen ob am Laufen = `sudo service ssh status`
 - * erstmal installieren, wenn nicht da = `sudo apt-get install openssh-server`
 - Server (neu) starten = `sudo service ssh restart`
 - * falls die Fehlermeldung “no hostkeys available” erscheint, diese ggf. neu erzeugen via
“`sudo ssh-keygen -A`”
 - Server stoppen = `sudo service ssh stop`
-

Verbindung herstellen

- `ssh` = SSH Client um Verbindung mit einem SSH Server aufzubauen
 - `ssh USERNAME@SERVERNAME`
 - * `USERNAME` ist optional; wenn fehlend wird der eigene Benutzername der aktuellen shell genommen, von der der Aufruf kommt
 - * `SERVERNAME` = IP (zB 192.168.1.32) oder Rechnername (herzblatt.gesucht.de)
 - wenn:
 - lokaler SSH Server gestartet wurde (siehe oben)
 - SSH key pair generiert und unter Standardnamen im `.ssh` Verzeichnis abgelegt wurden
 - lokale `$HOME/.ssh/authorized_keys` Datei mit public key erweitert wurde
 - dann funktioniert:
 - `ssh localhost` = **Verbindung zum eigenen Computer** (via SSH) :D
 - mit “`exit`” schliessen sie die Verbindung und kehren zur “ssh-aufrufenden Shell” zurück
-

Dateitransfer im Netz

Kopieren

`scp` - SSH-basierter cp-Befehl für Dateitransfer von und zu Servern

- `scp SOURCE TARGET`
 - `SOURCE` und `TARGET` können Serverangaben enthalten im Format: `USER@SERVER:FILEorFOLDER`
 - * “`USER@`” - optionaler Nutzernamen, ansonsten eigener Nutzernamen verwendet
 - * “`SERVER :`” - optionaler Servername (IP oder Adresse) im internen Netzwerk oder Internet; wenn nicht angegeben wird hier der eigene Rechner angenommen (`localhost`)
 - * “`FILEorFOLDER`” - Name der Quell-/Zielfeile bzw. Verzeichnis
 - * BEACHTEN: **absolute und relative Pfade** (bzgl. `$HOME` im remote server) möglich
 - `SOURCE` oder `TARGET` müssen auf anderem Server liegen, **nicht für lokales Kopieren** vorgesehen!
- ggf. Passwortabfrage (wenn SSH Server dies erlaubt)
- Wenn **pub-key Authentifizierung** eingerichtet ist (und der pub-key auf dem Server schon liegt), wird diese automatisch verwendet (keine Abfrage von login oder Passwort)
- “`-r`” - recursively - alle Unterverzeichnisse und -dateien werden entsprechend der wildcard-Angabe (mit `*` oder `?`) kopiert

- z.B. "`scp elrond@rivendell:missingRings.txt .`" holt die fehlende Liste zum Nachschauen ins aktuelle Verzeichnis (wenn die Authentifizierung klappt und der Server erreichbar ist)
-

Synchronisieren/Backup

`rsync` - Kopieren/Synchronisierung von Verzeichnissen für Archivierung und Spiegelung

- `rsync -a SOURCE TARGET`
 - `SOURCE` und `TARGET` **wie bei scp** definierbar
 - `/` **am Ende** von `SOURCE` = nur INHALT des Verzeichnisses synchronisieren (sonst auch Verzeichnisname im `TARGET` angelegt)
 - beide **können auf dem gleichen Rechner sein** (lokale Archivierung)
- **Standardverhalten** = alles was in `SOURCE` genannt ist (bzw. via wildcards gemappt wird) wird nach `TARGET` kopiert (und dortige Varianten ggf. ersetzt)
 - kann eingeschränkt werden auf
 - * nur geänderte Dateien (neuer/grösser/kleiner)
 - * fehlende Dateien/verzeichnisse
 - * ... (siehe man page)
- `-a` = archiving = Dateirechte etc. bleiben erhalten, im Detail eine **Kombination von**
 - `-t` = timestamp = erhalte Zeitstempel
 - `-r` = recursive
 - `-l` = Links werden als solche kopiert (nicht deren Zielfile)
 - `-p` = permissions erhalten
 - `-g` = group Rechte erhalten
 - `-o` = ownership erhalten
 - `-D` = special files erhalten
- `-z` = zip compressed = komprimierte Übertragung für schnelleren Synch von Text-basierten Dateien
- (`-e ssh`) = execute ... = führt **SSH** für die Erstellung der remote Verbindung aus = **Standard bei neueren rsync Versionen**
- `-nv` = no execution + verbose = Testlauf um zu prüfen was wohin synchronisiert werden wird
- `-u` = update = nur neuere Dateien ersetzt
- `-b` = backup = ältere Dateien im Ziel werden umbenannt

Achtung MacOSs ...

MacOs User sollten ihre [rsync Version aktualisieren](#), da die normal installierte Version aus lizenzrechtlichen Gründen veraltet ist!

> Tutorials <

Für einen besseren Überblick studieren sie doch bitte dieses

- [Online Tutorial zu rsync](#)
-

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](#) by [Dr. Eberle Zentrum für digitale Kompetenzen, Universität Tübingen](#)