

Secure Shell (SSH)

Dozent: Prof. Dr. Michael Eichberg
Kontakt: michael.eichberg@dhbw.de
Version: 1.4
Quellen: Teilweise basierend auf Folien von Prof. Dr. Henning Pagnia
bzgl. Netzwerksicherheit

Folien: [HTML] <https://delors.github.io/sec-ssh/folien.de.rst.html>
[PDF] <https://delors.github.io/sec-ssh/folien.de.rst.html.pdf>
Fehler melden: <https://github.com/Delors/delors.github.io/issues>

Secure Shell (SSH)

Verschlüsselte Verbindung

SSH ermöglicht die sichere Fernanmeldung von einem Computer bei einem anderen (typischerweise über TCP über Port 22). Es bietet mehrere Optionen für eine starke Authentifizierung und schützt die Sicherheit und Integrität der Kommunikation durch starke Verschlüsselung.

Ablauf

1. Authentisierung des Server-Rechners
2. Authentisierung des Benutzers (bzw. des Clients) mittels
 - a. Passwort
 - b. ~~.rhosts-Eintrag~~
 - c. privatem (RSA-)Key (hauptsächlich verwendete Methode)
3. Kommunikation über symmetrisch verschlüsselte Verbindung

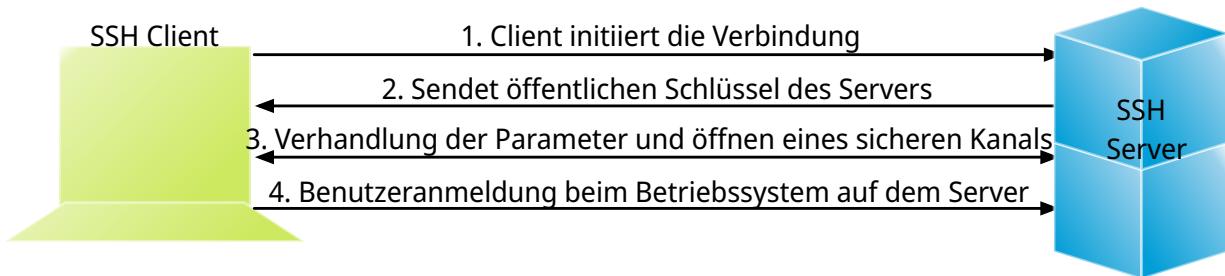
Die Authentifizierung mittels eines Schlüsselpaars dient primär der Automatisierung (dann wird auch keine „Schlüsselphrase“ zum Schutz des Passworts verwendet).

Auf jeden Fall ist effektives Schlüsselmanagement erforderlich:

[...] In einigen Fällen haben wir mehrere Millionen SSH-Schlüssel gefunden, die den Zugang zu Produktionsservern in Kundenumgebungen autorisieren, wobei 90 % der Schlüssel tatsächlich ungenutzt sind und für einen Zugang stehen, der zwar bereitgestellt, aber nie gekündigt wurde.

—SSH.com (Dez. 2023)

Secure Shell (SSH) - Protokoll



Beide Seiten haben einen Public-private Key Schlüsselpaar zur gegenseitigen Authentifizierung

User Keys: ■ Authorized keys - Serverseitige Datei mit den öffentlichen Schlüsseln der Nutzer

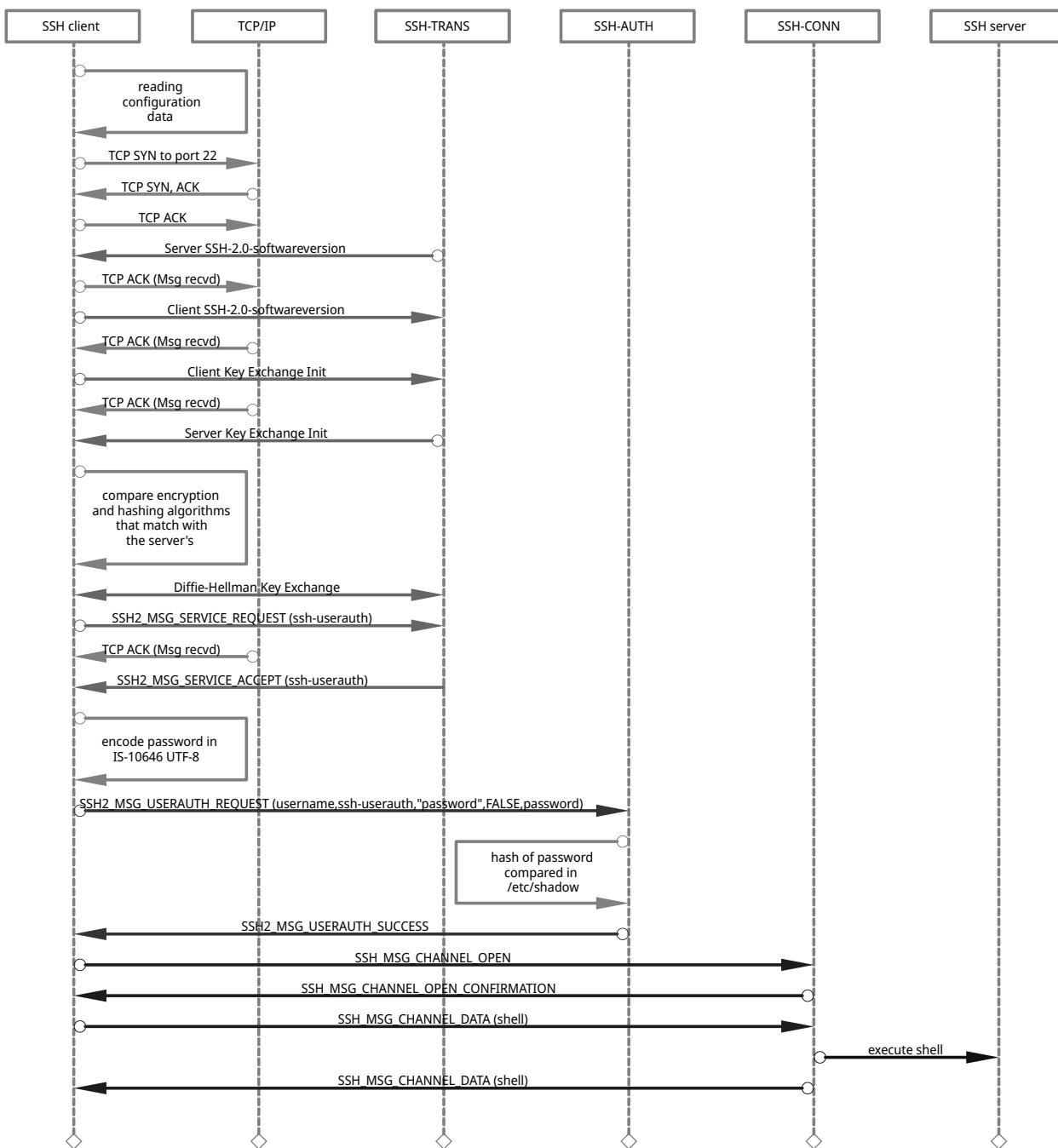
■ Identity keys - private Schlüssel der Nutzer

Host keys: dienen der Authentifizierung von Servern (verhindern Person-in-the-Middle-Angriffe)

Session Keys: werden für die symmetrische Verschlüsselung der Daten in einer Verbindung verwendet. Session Keys (Sitzungsschlüssel) werden während des Verbindungsbaus ausgehandelt.

Im Falle von SSH gibt es kein initiales Vertrauen zwischen Server und Client.

Secure Shell (SSH) - Authentifizierung mittels Passwort



CC 4.0 - Aleksey Valov - 29 February 2016

The font was changed to "Noto Sans Display" for better compatibility.

Secure Shell (SSH) - Verbindungsauflaufbau - Beispiel

```
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug1: Connecting to example.org [1.2.3.4] port 22.
debug1: Connection established.
debug1: identity file /home/user/.ssh/id_rsa type -1
debug1: identity file /home/user/.ssh/id_rsa-cert type -1
debug1: identity file /home/user/.ssh/id_dsa type -1
debug1: identity file /home/user/.ssh/id_dsa-cert type -1
debug1: Remote protocol version 1.99, remote software version OpenSSH_5.8
debug1: match: OpenSSH_5.8 pat OpenSSH*
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_5.5p1 Debian-6
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server→client aes128-ctr hmac-md5 none
debug1: kex: client→server aes128-ctr hmac-md5 none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
debug1: Host 'example.org' is known and matches the RSA host key.
debug1: Found key in /home/user/.ssh/known_hosts:1
debug1: ssh_rsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: Roaming not allowed by server
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password,keyboard-interactive,hostbased
debug1: Next authentication method: publickey
debug1: Trying private key: /home/user/.ssh/id_rsa
debug1: Trying private key: /home/user/.ssh/id_dsa
debug1: Next authentication method: keyboard-interactive
debug1: Authentications that can continue: publickey,password,keyboard-interactive,hostbased
debug1: Next authentication method: password
user@example.org's password:
debug1: Authentication succeeded (password).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: Sending environment.
debug1: Sending env LANG = en_US.UTF-8
```

Secure Shell - Unterstützte Transport-Verschlüsselungsverfahren

OpenSSH_10.2p1

Cipher Name	Status	Beschreibung
3des-cbc	DEPRECATED	three-key 3DES in CBC mode
aes256-cbc	LEGACY	AES in CBC mode, with a 256-bit key
aes192-cbc	LEGACY	AES with a 192-bit key
aes128-cbc	LEGACY	AES with a 128-bit key
aes128-ctr	RECOMMENDED	AES in CTR mode with a 128-bit key
aes192-ctr	RECOMMENDED	AES in CTR mode with a 192-bit key
aes256-ctr	RECOMMENDED	AES in CTR mode with a 256-bit key
aes128-gcm@openssh.com	RECOMMENDED	AES in Galois/Counter Mode (GCM) with a 128-bit key
aes256-gcm@openssh.com	RECOMMENDED	AES in Galois/Counter Mode (GCM) with a 256-bit key
chacha20-poly1305@openssh.com	RECOMMENDED	ChaCha20 stream cipher with Poly1305 MAC

DEPRECATED: Unterstützt, aber standardmäßig deaktiviert.

LEGACY: Unterstützt für Rückwärtskompatibilität, aber nicht empfohlen.

RECOMMENDED: Empfohlen für den Einsatz.

Anmerkung

CBC-Modi sind anfällig für bestimmte Angriffe (z. B. Padding-Oracle-Angriffe) und werden insbesondere für Rückwärtskompatibilität unterstützt.

Algorithmen unterstützt von älteren OpenSSH-Versionen

- blowfish-cbc , Blowfish in CBC mode
- twofish256-cbc , Twofish in CBC mode,with a 256-bit key
- serpent256-cbc , Serpent in CBC mode, with a 256-bit key
- arcfour , the ARCFOUR stream cipher with a 128-bit key
- idea-cbc , IDEA in CBC mode
- cast128-cbc , CAST-128 in CBC mode

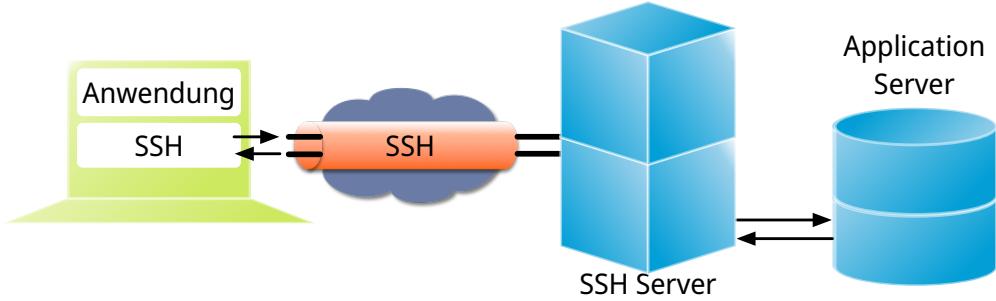
Secure Shell (SSH) - Risiken durch mangelnde Schlüsselverwaltung

- Schlüssel werden nicht regelmäßig ausgetauscht
- Schlüssel werden nicht gelöscht, wenn sie nicht mehr benötigt werden
- viele (die meisten) Schlüssel werden nicht verwendet
- Es ist oft nicht bekannt, wer Zugriff auf welche Schlüssel hat(te)
- Es ist nicht bekannt, welche Schlüssel auf welche Systeme Zugriff haben
- Malware kann SSH-Schlüssel stehlen
- SSH Keys können ggf. privilegierten Zugriff gewähren
- SSH Keys können benutzt werden, um Backdoors zu verstecken
- Server keys erlauben ggf. Person-in-the-Middle-Angriffe

SSH Tunneling

- ermöglicht die Übertragung beliebiger Netzwerkdaten über eine verschlüsselte SSH-Verbindung. z. B.
 - um ältere Anwendungen zu verschlüsseln.
 - um VPNs (Virtual Private Networks) zu implementieren.
 - um über Firewalls hinweg auf Intranetdienste zuzugreifen.
- ermöglicht auch Port-forwarding

(Lokale Ports werden auf entfernten Rechner weitergeleitet.)



SSH und „Back-tunneling“

- Der Angreifer richtet einen Server außerhalb des Zielnetzwerks ein
 - Nach Infiltration des Zielsystems verbindet der Angreifer sich von innen mit dem externen SSH-Server.
 - Diese SSH-Verbindung wird so eingerichtet, dass eine TCP-Port-Weiterleitung von einem Port auf dem externen Server zu einem SSH-Port auf einem Server im internen Netzwerk möglich ist.
 - Die meisten Firewalls bieten wenig bis gar keinen Schutz dagegen.
-

Es ist in diesem Fall besonders interessant für den Angreifer den SSH Server zum Beispiel bei einem Cloud-Anbieter zu betreiben, welcher von dem Unternehmen standardmäßig verwendet wird (am Anfang steht immer die Aufklärung!). In diesem Fall wird die Firewall keine ausgehenden SSH-Verbindungen dorthin blockieren.

Schwachstellen in SSH

Nearly 11 million SSH servers vulnerable to new Terrapin attacks

[...] It [The Terrapin attack] manipulates sequence numbers during the handshake process to compromise the integrity of the SSH channel, particularly when specific encryption modes like ChaCha20-Poly1305 or CBC with Encrypt-then-MAC are used. [...]

By Bill Toulas

—January 3, 2024 10:06 AM

SSH und Quantencomputer

[...] ssh(1): add a warning when the connection negotiates a non-post quantum key agreement algorithm.

—OpenSSH 10.1 Release Notes