Berner
Fachhochschule

Semesterarbeit

Sicherer Umgang mit dem SSH-Serverdienst von OpenSSH

Studiengang	CAS IT Security Management
Autor	Mauro Guadagnini
Experte	Hansjürg Wenger

Version 1.0 vom 24. September 2023

- Technik und Informatik
- Weiterbildung

Abstract

Das Ziel dieser Arbeit liegt darin, den SSH-Serverdienst von OpenSSH zu verstehen und abzusichern. Zudem stellt sich die Frage, wie sicher die Standardausführung der Server-Konfiguration zu betrachten ist und welche Parameter begründet anzupassen sind. In der Praxis wird der SSH-Server meist nur aktiviert und minimal angepasst, was sich als Sicherheitsrisiko herausstellen könnte. Daher gilt es für die Anwendungsfälle „Kommandozeilenzugriff“, „Dateiübertragungen“ sowie „Jumphost“ möglichst sichere Konfigurationen festzulegen und sämtliche Optionen zu beleuchten.

Um diese Fragen zu beantworten, werden die SSH-Protokoll-Spezifikationen betrachtet, die OpenSSH-Server-Parameter analysiert sowie Konfigurationsempfehlungen von diversen Publikationen und Behörden geprüft. Aus der Kombination der Empfehlungen resultiert ein Minimalstandard, der auf den SSH-Server in einer neuen Grundkonfiguration angewendet und mit dessen Standardkonfiguration verglichen wird.

Es stellt sich heraus, dass die OpenSSH-Server-Standardkonfiguration einen Kompromiss zwischen Sicherheit und Komfort bietet, welche es als schnellstmöglich abzusichern gilt. Die von OpenSSH begründete Wahl der eingesetzten Algorithmen ist gegebenenfalls durch eine vorgegebene Auswahl zu ersetzen, entspricht jedoch dem Stand der Technik (mit Ausnahme der RSA-Minimal-Schlüsselgrösse, welche es zu erhöhen gilt). Mittels reiner Passwort-Authentisierung kann eine SSH-Verbindung hergestellt werden, womit dessen Sicherheit auf derer des Benutzerkennworts beruht. Kaum eingeschränkte Forwarding- bzw. Weiterleitungs-Optionen ermöglichen das Etablieren von gegebenenfalls ungewollten Datenflüssen über einen SSH-Server.

Die erarbeitete und verifizierte Grundkonfiguration schliesst ungewollte Funktionalitäten, implementiert eine Multi-Faktor-Authentisierung und ermöglicht einen gezielten Einsatz der genannten Anwendungsfälle, wobei nur das Nötigste geöffnet wird. Weitere Sicherheitserhöhungen werden durch zusätzliche Varianten wie unter anderem der Einsatz von Zertifikaten mit CA (Certificate Authority) sowie FIDO2-Authentisierung mit einem YubiKey ermöglicht.

Titelbild: Illustration zu OpenBSD Version 4.3 [1], automatisiert hochskaliert und editiert durch Upscale.media

Inhaltsverzeichnis

1. Einleitung

Dieses Dokument behandelt den Umgang mit dem SSH-Serverdienst von OpenSSH, um begründete und möglichst sichere Konfigurationen für spezifische Anwendungsfälle zu ermitteln.

Hierbei handelt es sich um eine Semesterarbeit im CAS IT Security Management (2023 FS) an der Berner Fachhochschule gemäss einem selbst gewählten und bewilligten Projektantrag. Ausgangslage und Zielsetzung wurden entsprechend Vorgaben und Vorstellungen des Autors definiert.

1.1. Ausgangslage

OpenSSH ist gemäss Praxiserfahrung die am weitest verbreitete Implementation des SSH-Protokolls und in einer Vielzahl von Produkten wie unter anderem BSD-, Linux- und Windows-Betriebssystemen sowie diverse Firewalls und Switches anzutreffen [2].

Dabei bietet OpenSSH reichlich Konfigurationsmöglichkeiten (zum Beispiel hat dessen SSH-Serverdienst über 90 Parameter [3]), welche in der Praxis meist aus Zeitgründen nicht beachtet und auf den gegebenenfalls vordefinierten Einstellungen belassen werden. Im Internet existieren diverse Einstellungsempfehlungen, die oft nicht alle Parameter berücksichtigen und/oder ihre Wahl nicht begründen [1]. Daher stellt sich die Frage, ob die daraus resultierenden Konfigurationen als einigermassen sicher betrachtet werden können oder ob da noch Potential zur Sicherheitserhöhung vorhanden ist.

SSH kann für den Kommandozeilenzugriff, aber u. a. auch für Dateiübertragungen oder Weiterleitungen verwendet werden, wofür jeweils unterschiedliche Einstellungen geeignet sind.

1.2. Zielsetzung

OpenSSH wird primär für das Betriebssystem OpenBSD entwickelt und wird für andere Produkte als „portable“ Version bereitgestellt, welche in ihrer Versionsbezeichnung mit einem „p“ markiert wird [4]. Mit Fokus auf die Implementation in Version 9.3 vom 19. Juli 2023 in OpenBSD 7.3 vom 10. April 2023 gilt es folgende Ziele zu erreichen:

- Verstehen des SSH Protokolls und den Optionen von OpenSSH, primär dessen SSH-Serverdienst
- Aufbau und Verifikation möglichst sicherer Konfigurationen zu folgenden Anwendungsfällen:
  - Kommandozeilenzugriff
  - Dateiübertragungen
  - „Jumphost“ (Verwendung als Zwischenrechner)
- Agent Forwarding einsetzen

Optional sind die Standardausführungen und Versionen diverser Produkte (siehe Kapitel 2.1, Listenpunkt 6) zu Vergleichen und Bewerten.

Aus den definierten Zielen werden Massnahmen und Arbeitsflüsse zur Einrichtung und dem Umgang mit der SSH-Implementation von OpenSSH ermittelt und empfohlen, um dessen Einsatz mit einem möglichst geringen Restrisiko gewährleisten zu können.

Im Rahmen dieser Arbeit werden Betriebssysteme in Virtuellen Maschinen (VMs) betrieben. Der Aufbau wird so beschrieben, dass er mit der entsprechenden Soft- und Hardware selbstständig nachvollziehbar ist.

[1] Von blindem Kopieren von Code oder Konfigurationen ist aufgrund unberechenbarer Konsequenzen abzuraten

2. Planung

2.1. Vorgehen

Diese Arbeit ist entsprechend folgendem Vorgehen aufgebaut:

1. Beginn
   a) Ausgangslage etablieren
   b) Ziele definieren
2. Planung
   a) Überblick zur Verfügung stehende Mittel inkl. Beschaffung zusätzlicher Komponenten
   b) Definition Massnahmen zur Zielerarbeitung
   c) Kontrolle Planung gemäss Zielsetzung
3. Recherche
   a) SSH-Protokoll
   b) Parameter OpenSSH Implementation
   c) Konfigurationsempfehlungen / „Best Practices“
   d) Aktuelle Schwachstellen bzw. Verwundbarkeiten OpenSSH Versionen
4. Aufbau
   a) OpenSSH Konfigurationen zu vorgehend erwähnten Anwendungsfällen (siehe Kapitel 1.2)
   b) Agent Forwarding
5. Verifikation
   a) Verifizieren des Aufbaus gemäss Zielsetzung
   b) Sicherstellung keiner unbeabsichtigten Datenflüsse
6. (Optional) Vergleich Standardausführungen und Versionen folgender Produkte
   - FreeBSD 14
   - Linux-Distributionen: Debian 12, Red Hat Enterprise Linux 9 & Ubuntu 22.04 LTS
   - VMware vSphere 8 (vCenter und ESXi Hypervisor)
   - Windows Server 2022
7. Abschluss
   a) Fazit, Rückblick und Ausblick definieren
   b) Dokumentation finalisieren

Zur Ermöglichung der Reproduktion werden zugehörige Befunde und Vorgehen dokumentiert.

2.2. Verfügbare Komponenten

Folgende Komponenten stehen für diese Arbeit zur Verfügung:

Komponente	Hersteller	Bezeichnung	Version	Datum	Quelle
Betriebssystem	OpenBSD Team	OpenBSD	7.3	10.04.2023	[5]
Hypervisor	Oracle	VirtualBox	7.0.8	18.04.2023	[6]
Betriebssystem	Debian Project	Debian	5.0.8	22.01.2011	[7]
Software-Suite	OpenSSH Team	OpenSSH	9.3	19.07.2023	[5]
Firewall	Rubicon Communications LLC (Netgate)	pfSense	2.7.0 CE	29.06.2023	[8]

Tabelle 2.1.: Verfügbare Komponenten

Die alte Debian-Version wird zum Aufbau eines Servers mit veraltetem Software-Stand im Testfall in Kapitel 5.5.2 verwendet.

2.3. Massnahmen zur Zielerarbeitung

Entsprechend der Zielsetzung aus Kapitel 1.2 und dem Vorgehen aus Kapitel 2.1 ergeben sich folgende Massnahmen zur Zielerarbeitung:

1. Nachvollziehen des SSH-Protokolls
2. OpenSSH-Serverdienst `sshd` Konfigurationsparameter analysieren
3. Recherche Empfehlungen und „Best Practices“
4. Recherche bekannter Schwachstellen in OpenSSH
5. Erörtern anzupassender OpenSSH-Optionen für die Anwendungsfälle aus Kapitel 1.2 und Durchführen entsprechender Implementationen
   - Kommandozeilenzugriff
   - Dateiübertragungen
   - „Jumphost“ (Verwendung als Zwischenrechner)
6. Agent Forwarding analysieren und implementieren
7. Definition von Arbeitsflüssen zur Einrichtung und Umgang mit OpenSSH inkl. Vorgehen bei notwendigem Ersetzen eines Schlüssels (z.B. bei Verlust oder Kompromittierung)
8. Erarbeitung Testfälle zur Verifikation der Implementationen
9. Überprüfen der Konfigurationen gemäss definierter Testfälle
10. (Optional) Standard-Installation, Analyse und Vergleich der Standardausführungen von Produkten aus Kapitel 2.1, Punkt 6

3. Recherche

3.1. SSH (Secure Shell)

SSH (Secure Shell) ist ein Protokoll, welches einen sicheren, verschlüsselten Kanal über ein unsicheres Netzwerk ermöglicht („Secure Channel“) [9]. Die Webseite zu OpenSSH verweist auf die implementierten Spezifikationen [10]. Dieses Kapitel bietet eine Zusammenfassung des SSH-Protokolls. Eine vertiefte Ausführung kann den jeweils referenzierten Quellen entnommen werden. Es wird die Version 2.0 des SSH-Protokolls (auch bekannt als „SSH-2“) behandelt. Da diese Version bereits seit 2006 als Standard [11] existiert (und in OpenSSH seit Juni 2000 [12] implementiert ist), wird auf die älteren Versionen nicht eingegangen und von deren Gebrauch abgeraten. SSH-Protokoll Version 1 wurde aus OpenSSH mit Version 7.6 vom 3. März 2017 entfernt [13].

Das SSH-Protokoll besteht hauptsächlich aus 3 Komponenten bzw. Protokollen[2] [11]:

1. **Transport Layer Protocol** [14]
   Zur Server Authentisierung, bietet Vertraulichkeit und Integrität mit Perfect Forward Secrecy (beim Schlüsselaustausch mit Diffie–Hellman Key Exchange)
2. **User Authentication Protocol** (auch Authenication Protocol) [15]
   Authentisiert den Client beim Server und verwendet das Transport Layer Protocol
3. **Connection Protocol** [16]
   Multiplexing mehrerer logischer Streams/Kanäle über den verschlüsselten Tunnel und verwendet das User Authentication Protocol

Der Client sendet eine Service-Anfrage nach dem Herstellen einer Transport-Layer-Verbindung und eine Weitere nach dem Vollenden der „User Authentication“, was die Definition und Koexistenz weiterer „Connection“-Protokolle ermöglicht [11]. Das Connection Protocol bietet mit dessen Kanälen diverse Einsatzmöglichkeiten (u. a. Shell-Sessions bzw. Kommandozeilenzugriff und Forwarding von TCP/IP-Ports) [11].

Da der Server vor der Authentisierung bereits prozessor- und arbeitsspeicher-intensive Arbeiten ausführt, ist das Protokoll anfällig für DoS (Denial of Service) Attacken [11].

[2] Die Namen der Protokolle beinhalten häufig die englische Bezeichnung „Protocol“, weshalb in diesem Dokument auch der englische Begriff verwendet wird

3.1.1. Keys / Schlüssel

Ein Server mit Host-Keys muss mindestens über einen Schlüssel für jeden im DSS (Digital Signature Standard) vorgegeben Algorithmus verfügen: **RSA**, **ECDSA** und **EdDSA** [11][17]. Der Algorithmus DSA wird in der aktuellsten Version des DSS nicht mehr verwendet [17]. Bei diesen Schlüsseln handelt es sich um Private-Keys.

Damit der Client die Verbindung zum korrekten Server verifizieren kann, muss ihm der zugehörige Public-Key bekannt sein (mittels lokaler Datenbank des Clients oder über eine hinterlegte CA (Certificate Authority)) [11].

Sollte der Client keine Kenntnis über den Public-Key verfügen, so kann dieser dennoch eine Verbindung zum Server herstellen, wobei eine solche Übertragung verwundbar gegen Man-in-the-Middle-Angriffe ist und normalerweise nicht erlaubt werden sollte [11]. Möchte man mit OpenSSH eine Verbindung herstellen, ohne zuvor den zugehörigen Public-Key zu wissen, so erscheint eine entsprechende Warnung, bei welcher man die Verbindung ggf. verifizieren und bestätigen kann.

Eine Möglichkeit, um dem Client den Public-Key „out-of-band“ mitzuteilen, wäre die Publikation des Public-Keys mittels DNS-Eintrag des Typs „SSHFP“ (inkl. Absicherung mittels DNSSEC), welcher beim Verbindungsaufbau geprüft und verifiziert werden kann [18].

3.1.2. Algorithmen und Formate

Alle von SSH verwendeten kryptografischen Algorithmen sind bekannt und verbreitet, wobei vernünftige Schlüssel-Grössen jahrzehntelangen Schutz gegen kryptoanalytische Angriffe bieten [11]. Sollte ein Algorithmus gebrochen werden, kann der Wechsel zu einem Anderen ohne Modifikation des Basis-Protokolls vollzogen werden [11]. Spezifische Verschlüsselungsverfahren (ausser in der Industrie etablierten und akzeptierten Verfahren) werden in den SSH-Spezifikationen nicht empfohlen [11].

Das SSH-Protokoll erlaubt die Verhandlung von Algorithmen und Formaten zur Verschlüsselung, Integritätssicherung, Schlüsselaustausch, Kompression und Public-Keys, wobei die Verwendung eigener Algorithmen und Formate möglich ist und deren Identifikation einem Namen mit spezifischen Textformat entspricht [11].

Es gibt Standard-Algorithmen und -Methoden, die eine SSH-Implementation unterstützen muss sowie auch eine Auswahl optionaler Algorithmen und Methoden [11]. Hierbei gibt es zwei Formate für deren Bezeichnungen [11]:

- Namen ohne At-Symbol „@“
  sind zur Zuweisung durch die IETF (Internet Engineering Task Force) reserviert, wobei sie zuvor bei der IANA (Internet Assigned Numbers Authority) registriert sein müssen
- Namen mit At-Symbol „@“
  sind für zusätzliche Algorithmen und Methoden verwendbar, wobei das At-Symbol „@“ nur ein Mal vorkommen darf, der Teil vor dem „@“ ein beliebiger Name sein kann und der Teil nach dem „@“ ein gültiger FQDN sein muss

Sämtliche Namen müssen „case-sensitive“, im druckfähigen US-ASCII-Format und nicht länger als 64 Zeichen sein [11]. Das Verschlüsselungsverfahren mit Namen „none“ [19] existiert nur zu Debugging-Zwecken und sollte auch nur dafür verwendet werden (gleiches gilt für den MAC (Message Authentication Code) „none“) [11].

3.1.3. Transport Layer Protocol

Das SSH Transport Layer Protokoll bietet einen einmaligen Session-Identifikator („Session-ID“), welcher von höheren Protokollen zur Verknüpfung mit einer Session verwendet werden kann, um Replay Attacken zu verhindern [11]. Die Generierung der Session-ID beinhaltet pseudo-zufällige Daten des Algorithmus und Schlüsselaustauschprozesses, womit die Wahrscheinlichkeit für zwei gleiche Session-IDs minimal ausfällt [11].

Das Protokoll wird typischerweise auf TCP/IP unter TCP-Port 22 betrieben und kann als Basis für verschiedene sichere Netzwerkdienste verwendet werden [14].

Beim Verhandeln des Algorithmus wird eine Liste möglicher Methoden für den Schlüsselaustausch gesendet, wobei die erste Methode die Bevorzugte ist und die Liste nach kryptografischer Stärke (mit dem stärksten Eintrag zuerst) sortiert sein sollte [11]. Konkret werden in diesem Protokoll die Schlüsselaustauschmethode und Algorithmen zu Kompression, Public-Key, zur symmetrischen Verschlüsselung, MAC und Hashing verhandelt [14].

Die Authentisierung in diesem Protokoll erfolgt rein Host-basiert (ohne Benutzerauthentisierung, welche überliegenden Protokollen überlassen wird) [14]. Beide Seiten müssen einen Identifikations-String mit maximal 255 Zeichen senden, welcher die Protokollversion (hier „2.0“) und Software-Version beinhalten muss (Optional können Kommentare mitgegeben werden) [14]. Nach dem Senden der Identifikation wird mit dem Schlüsselaustausch begonnen.

Ein Paket des SSH Transport Layer Protokolls entspricht folgendem Format [14]:

- `packet_length` [4 Bytes]
  Paketlänge in Bytes ohne `packet_length` und ohne `mac`
- `padding_length` [1 Byte]
  Padding-Länge in Bytes
- `payload` [$n1$ Bytes]
  Paketinhalt [$n1$ = `packet_length` - `padding_length` - 1]
  Bei aktivierter Kompression ist der Inhalt komprimiert, zu Beginn ist diese jedoch immer aus (der Inhalt wird nach der Kompression verschlüsselt)
- `random_padding` [$n2$ Bytes]
  Zufällig generierter Padding-Inhalt [$n2$ = `padding_length`, min 4 Bytes, max 255 Bytes]
  Muss mit den vorherigen Feldern zusammen ein Multiples der Verschlüsselungs-Block-Grösse oder mindestens 8 Bytes ergeben
- `mac` [m Bytes]
  MAC (Message Authentication Code) [m = Grösse MAC]
  Enthält entsprechende MAC-Bytes sobald dieser verhandelt wurde (zu Beginn leer)

Sobald der Verschlüsselungsalgorithmus definiert wurde, werden alle zuvor erwähnten Paketfelder (ausser `mac`) verschlüsselt [14].

3.1.3.1. Verschlüsselung

Im RFC 4253 des Transport Layer Protokolls [14] ist 3DES-CBC (`3des-cbc`) als einzig notwendiger („required“) Algorithmus hinterlegt, wobei die Verschlüsselung mittels 3DES nach 2023 vom NIST (National Institute of Standards and Technology) untersagt wird [20]. Dieser ist aus historischen Gründen und zwecks Interoperabilität noch aufgeführt [14]. Der einzige als empfohlene (nicht optionale) aufgeführte Algorithmus ist AES mit einem 128-Bit-Schlüssel im CBC Modus (`aes128-cbc`) [14]. Die Verwendung von SHA-1 wird vom NIST und RFC 9142 nicht mehr empfohlen [20][21].

Weitere Verschlüsselungsmodi werden mit RFC 4344 aufgeführt, welche neue Verschlüsselungsalgorithmen im CTR Modus empfehlen, nämlich `aes128-ctr` (AES mit einem 128-Bit-Schlüssel) und `aes192-ctr` sowie `aes256-ctr` mit entsprechenden Schlüsselgrössen und zusätzlich `3des-ctr` [22]. Es wird einem die Verwendung von `aes128-ctr` sehr nahegelegt sowie auch auf die Verwundbarkeit der ursprünglich aufgeführten Algorithmen im CBC Modus gegen Chosen-plaintext Attacken hingewiesen [22].

3.1.3.2. MAC (Message Authentication Code)

Der MAC wird anhand eines „Shared Secret“, einer Sequenznummer und dem unverschlüsselten Paket (ohne `mac`) generiert [14]. Der RFC 4253 des Transport Layer Protokolls listet als einzig notwendiger („required“) MAC-Algorithmus `hmac-sha1` auf, wobei Verschlüsselungen mit HMAC mit einer Schlüssellänge kleiner als 112 Bits gemäss NIST nicht mehr erlaubt sind [20].

HMAC-SHA2-Algorithmen `hmac-sha2-256` und `hmac-sha2-512` werden mit RFC 6668 hinzugefügt [23].

3.1.3.3. Schlüsselaustauschmethoden

Als ursprünglich notwendige („required“) Schlüsselaustauschmethoden wurden folgende zwei definiert:

- `diffie-hellman-group1-sha1`
  DHKE mit „Oakley“-Gruppe 2 (Identifikationsnummer 2, 1024 Bits)[3] gemäss RFC 2409 [24]
  Verwendung durch RFCs 8270 und 9142 abgeraten [25][21]
- `diffie-hellman-group14-sha1`
  DHKE mit Gruppe mit Identifikationsnummer 14 und 2048 Bits gemäss RFC 3526 [26]

Gruppe 1 verfügt über eine Stärke entsprechend einer symmetrischen Verschlüsselung mit ungefähr 70 bis 80 Bits [26]. Von beiden Gruppen ist Gruppe 14 von der NIST genehmigt [27].

Mit der Erweiterung durch RFC 4419 sind folgende neuen Methoden dazu gekommen [28]:

- `diffie-hellman-group-exchange-sha1`
  Diffie-Hellman Group and Key Exchange mit SHA-1, wessen Verwendung in RFC 9142 wieder abgeraten wird [21]
- `diffie-hellman-group-exchange-sha256`
  Diffie-Hellman Group and Key Exchange mit SHA-256

Diese beiden Methoden ermöglichen den DHKE mit neuen, vom Server vorgeschlagenen Gruppen und benötigen keine zuvor fixierten Werte [28].

Seit RFC 8270 beträgt die kleinste empfohlene Gruppen-Grösse für Diffie-Hellman 2048 Bits und könnte zukünftig bei 3072 Bits liegen [25].

[3] Obwohl in der Bezeichnung „Group1“ steht, wird die Gruppe mit der ID 2 verwendet [14]

Betreffend ECC (Elliptic Curve Cryptography) wird mit RFC 5656 ECDH (Elliptic Curve Diffie-Hellman) key exchange eingeführt, wessen Methoden den Namen `ecdsa-sha2-[identifier]` tragen, wobei `[identifier]` für die Domain-Parameter der elliptischen Kurve stehen [29].

Notwendige („required“), genannte Kurven sind `nistp256`, `nistp384` und `nistp521` (auch bekannt als `secp256r1`, `secp384r1` und `secp521r1` [30]), welche in SEC2 [31] definiert sind [29]. Empfohlene Kurven können im zuvor erwähnten RFC-Dokument gefunden werden [29]. Zum Beispiel lautet das Format für ECDH mit SHA-2 und der elliptischen Kurve `nistp256`: `ecdh-sha2-nistp256` [29].

Optional wird zusätzlich der Algorithmus ECMQV (Elliptic Curve Menezes-Qu-Vanstone) key exchange aufgeführt (`ecmqv-sha2`) [29].

Mit RFC 8731 werden neue Methoden für die DHKE-Funktionen X25519 mit SHA-256 (`curve25519-sha256`) und X448 mit SHA-512 (`curve448-sha512`) unterstützt [32].

Dadurch, dass bei der ECC kleinere Schlüssel-Grössen für dieselbe Sicherheitsstufe wie zum Beispiel RSA nötig sind, erlaubt die ECC eine effizientere Implementation [29].

3.1.3.4. Public-Keys

Für Public-Keys ist allein das Format `ssh-dss` (DSA gemäss DSS mit SHA-1) „required“, wobei das RSA-Format `ssh-rsa` zusätzlich empfohlen wird [14]. Optional können Zertifikate zur Autorisierung mitgegeben werden.

Mittels RFC 5656 können Formate der ECC (Elliptic Curve Cryptography) mit dem Algorithmus ECDSA und SHA-2 genutzt werden, welche analog zum vorherigen Abschnitt „Schlüsselaustauschmethoden“ den Namen `ecdsa-sha2-[identifier]` tragen. Notwendige („required“), benannte sowie empfohlene Kurven sind hier dieselben wie in Kapitel 3.1.3.3 [29]. Zum Beispiel lautet das Format für ECDSA mit SHA-2 und der elliptischen Kurve `nistp256`: `ecdsa-sha2-nistp256` [29].

Gemäss RFC 8332 kann zusätzlich der Algorithmus RSA mit SHA-256 (`rsa-sha2-256`) oder SHA-512 (`rsa-sha2-512`) verwendet werden [33]. RFC 9142 ratet vom Verwenden von SHA-1 und RSA mit einem Schlüssel von 1024 Bit ab [21].

Mit RFC 8709 wurden die Algorithmen Ed25519 (`ssh-ed25519`) und Ed448 (`ssh-ed448`) etabliert [34].

3.1.3.5. Schlüsselaustausch („Key exchange“)

Beim Schlüsselaustausch sendet jede Seite eine Liste unterstützter Algorithmen mit Angabe eines präferierten Algorithmus pro Kategorie (u. a. Schlüsselaustauschmethoden, MAC und Verschlüsselungsalgorithmen) [14]. Der präferierte Algorithmus der anderen Seite kann durch vorgängiges senden eines entsprechenden „Initial Key Exchange“-Pakets erraten werden, wobei es bei einer falschen Wahl ignoriert und ein neues Paket versandt wird [14].

Als Ergebnis entsteht ein gemeinsamer Geheimwert `K` und ein sogenannter „Exchange Hash“ `H`, anhand derer diverse Schlüssel für die Verbindung berechnet werden [14]. `H` wird zusätzlich als eindeutiger Session-Identifikator für die Verbindung verwendet und wird auch bei einem späteren, erneuten Schlüsselaustausch nicht gewechselt [14].

Wird ein erneuter Schlüsselaustausch beantragt, wird dieser mit der bestehenden Verschlüsselung durchgeführt, wobei die Algorithmen-Wahl angepasst werden könnte [14]. Ein solcher Neu-Austausch wird nach jedem übertragenen Gigabyte oder spätestens nach einer Stunde der Verbindungsdauer empfohlen, was jedoch nur für Block-Grössen gilt, die kleiner als 128 Bit sind [14][22]. Bei Block-Grössen L mit 128 Bit oder mehr wird zu einem Neu-Austausch vor dem Versand von $2^{(L/4)}$ Blöcken geraten [22]. Bei grösseren Block-Grössen L (z.B. 256 Bit) sollte spätestens vor dem Versand oder vor dem Empfang von 2^{32} Blöcken neue Schlüssel generiert werden [22].

Für den Austausch weiterer Informationen während des Schlüsselaustauschs wird ein „Extension Negotiation“-Mechanismus ermöglicht, mit welchem u. a. Algorithmen für die Public-Key-Authentisierung gemäss Kapitel 3.1.4.1 oder ob Parameter zur Flusskontrolle berücksichtigt werden sollen („no-flow-control“) mitgeteilt werden können [35].

3.1.3.6. Service

Der Client beantragt nach dem Schlüsselaustausch einen Service mittels Namen, wobei `ssh-userauth` sowie `ssh-connection` reserviert sind und auch eigene Services mit einem Namensformat gemäss Kapitel 3.1.2 verwendet werden können [14]. Lehnt der Server den Service-Request ab, wird die Verbindung beendet [14]. Beim Service-Start kann dieser den zuvor erzeugten Session-Identifikator verwenden [14].

3.1.4. User Authentication Protocol

Beim User Authentication Protocol (auch SSH Authenication Protocol) wird davon ausgegangen, dass eine sichere Transport Layer Verbindung mit authentisiertem Server, verschlüsseltem Kommunikationskanal und Session-ID zur Verfügung steht [11]. Verschiedene Authentisierungsmethoden sind erlaubt, wobei ein Server nach wiederholten erfolglosen Versuchen seine Antworten verzögern soll, um die Schlüsselsuche für Angreifer schwieriger zu gestalten (20 Versuche pro Session [15]) [11].

Das Protokoll wird mit dem Service-Namen `ssh-userauth` über das SSH Transport Layer Protocol betrieben und verwendet den dort ermittelten Session-Identifikator („Exchange Hash“ `H` vom ersten Schlüsselaustausch) [15].

Der Server teilt dem Client die zulässigen Authentisierungsmethoden mit, welche dieser beliebig verwenden kann [15].

Ein „Authentication Request“-Paket des SSH Authentication Protokolls entspricht folgendem Format [15]:

- Message-ID `SSH_MSG_USERAUTH_REQUEST` (ID 50 [9]) [1 Byte]
- Benutzername [String]
- Service-Name [String]
  Bezeichnet den nach der Authentisierung zu startenden Service
- Methodenname [String]
  Enthält einen der folgenden Methodennamen:
  - `publickey`
    Public-Key-Authentisierung
    Einzig notwendige bzw. als „required“ gekennzeichnete Methode
  - `password`
    Passwort-Authentisierung
  - `hostbased`
    Host-basierte Authentisierung
  - `keyboard-interactive` [36]
    Interaktive Authentisierung mittels Tastatur (oder ähnlichem Gerät)
  - `none`
    Keine Authentisierung (darf vom Server nicht als unterstützte Methode aufgelistet werden)
- Weitere, methodenspezifische Felder

Weist der Server die Anfrage zurück, antwortet dieser mit einer Liste möglicher Methoden [15]. Bei einer Authentisierung mit mehreren Methoden teilt der Server beim Erfolg mit einer Methode ebenfalls eine solche Liste (ohne den bereits erfolgreichen Eintrag) mit [15]. Beide dieser Fälle treten mit der Message-ID `SSH_MSG_USERAUTH_FAILURE` (ID 51 [9]) auf [15].

Bei einer erfolgreichen Authentisierung mit allen benötigten Methoden antwortet der Server mit einer Meldung mit Message-ID `SSH_MSG_USERAUTH_SUCCESS` (ID 52 [9]), nach welcher der angeforderte Service gestartet wird [15].

Zusätzlich kann während des Authentisierungsvorgangs der Server eine sogenannte „Banner“-Nachricht mit Message-ID `SSH_MSG_USERAUTH_BANNER` (ID 53 [9]) senden, welche z.B. gegebenenfalls gesetzlich vorgegebene Warntexte beinhalten kann [15]. Der Client sollte diese Nachricht standardmässig darstellen, kann das Anzeigen solcher Nachrichten jedoch deaktivieren [15].

3.1.4.1. Public-Key-Authentisierung

Mit dieser Methode wird der Besitz eines bestimmten Private-Keys zur Authentisierung verwendet, wobei eine damit erstellte Signatur und der zugehörige Public-Key mittels Request-Paket gesendet wird [15]. Um eine unnötige Belastung der Ressourcen zu vermeiden, wird vor dem Signieren eine Bestätigung zum zu verwendenden Algorithmus eingeholt [15].

Die Signatur wird mit dem Private-Key u. a. anhand folgender Merkmale erstellt: Session-Identifikator, Benutzername, Service-Name, Public-Key [15].

Der Server prüft dann ob der mitgeteilte Schlüssel akzeptiert werden kann und wertet bei einem positiven Ergebnis die Signatur aus [15]. Ist diese zusätzlich zum gültigen Public-Key korrekt, so ist die Authentisierungsmethode erfolgreich abgeschlossen [15].

3.1.4.2. Passwort-Authentisierung

Bei der Passwort-Authentisierung wird das Passwort im Klartext und UTF-8-Encodierung gemäss RFC 3629 [37] im Request-Paket über den verschlüsselten Transport Layer gesendet, wobei die Interpretation und Validierung dem Server überlassen wird [15].

Der Server antwortet je nachdem mit einer positiven oder negativen Meldung [15]. Sofern implementiert, kann der Server das Ablaufen eines Passworts mitteilen und der Client sein Passwort ändern [15].

3.1.4.3. Host-basierte Authentisierung

Diese Methode verwendet den Private-Key des Client-Hosts zur Erstellung einer Signatur ähnlich wie bei der Public-Key-Authentisierung, wobei hier der Server nur die Client-Host-Identität prüft und danach keine weitere Authentisierung mehr stattfindet [15]. Daraufhin wird die Autorisierung gemäss Benutzer- und Client-Host-Namen durchgeführt [15].

Der RFC zum Authentication Protocol [15] bezeichnet diese Methode als ungeeignet für Hochsicherheitsumgebungen und empfiehlt zusätzlich das Verifizieren der Netzwerk-Adresse des Clients [15].

3.1.4.4. Keyboard-Interactive

Das Authentication Protocol wurde mit dem RFC 4256 um einen weitere Authentisierungsmethode erweitert, welche den Namen `keyboard-interactive` trägt [36]. Diese Methode erlaubt es die Authentisierungsmechanismen zu abstrahieren, sodass der Client keine Kenntnis des auf dem Server verwendeten Mechanismus benötigt und lediglich die Daten zur Authentisierung über eine Tastatur (oder ein ähnliches Gerät) eingegeben werden müssen [36]. Somit können zum Beispiel „One Time Password“-Mechanismen unterstützt werden [36].

3.1.5. Connection Protocol

Beim Connection Protocol wird von sicheren bzw. nicht kompromittierten Endpunkten ausgegangen [11]. Mittels „Proxy Forwarding“ oder „Reverse Proxy Forwarding“ können andere Protokolle wie z.B. HTTP über das SSH-Protokoll weitergeleitet werden [11]. X11-Display-Forwarding ist ebenfalls möglich, um grafische Anwendungen auf dem Server über SSH auf dem Client anzeigen zu können [11].

Das Protokoll wird mit dem Service-Namen `ssh-connection` auf den Transport Layer und User Authentication Protokollen betrieben [16]. Es bündelt „Channels“ bzw. Kanäle für u. a. Kommando-Ausführungen oder weitergeleitete TCP/IP-Verbindungen in einen einzelnen, verschlüsselten Tunnel [16]. Es gibt sogenannte „Global Requests“, welche den allgemeinen Status des entfernten Endpunktes beeinflussen (z.B. den Request zum TCP/IP-Forwarding über einen bestimmten Port) und von Clients sowie Server gesendet werden können [16].

Jede Terminal-Session, weitergeleitete Verbindung, etc. wird als Channel gehandhabt, wobei jede Seite einen Channel öffnen kann [16]. Sie werden mittels Nummer am Ende referenziert, die jedoch auf beiden Seiten unterschiedlich ausfallen kann [16]. Kanäle sind flussgesteuert („flow-controlled“), was bedeutet, dass Daten zu einem Kanal nur gesendet werden, wenn zuvor eine genügend grosse „Fenster-Grösse“ mitgeteilt wurde [16].

Bei einem Channel-Aufbau wird u. a. dessen Channel-Typ mittels Namen (z.B. `session`, `forwarded-tcpip` [9]) und ggf. Channel-spezifischen Werten angefordert [16]. Bei einem erfolgreichen Kanal-Aufbau kann dieser nach Gebrauch von beiden Seiten beendet werden [16].

3.1.5.1. Session

Als Session (Channel-Typ `session`) wird hier die entfernte Ausführung eines Programms (z.B. Shell oder Applikation) gehandhabt [16]. Für Sessions können Pseudo-Terminals in gewünschten Höhen und Breiten verwendet sowie auch das Öffnen von X11-Kanälen (Channel-Typ `x11`) angefragt werden [16]. In einer Session können Shells, Befehle oder Subsysteme (z.B. ein Dateiübertragungsmechanismus) gestartet werden, wobei eine Anfrage pro Channel durchgeführt wird [16].

3.1.5.2. TCP/IP Port Forwarding

Kanäle zum Weiterleiten von TCP/IP-Ports (Channel-Typ `forwarded-tcpip`) können verwendet werden, um Pakete an Hosts weiterzuleiten, wobei der Empfänger des Channel-Requests über das Erlauben der Verbindung bestimmt [16].

Channels dieses Typs sind unabhängig von Sessions [16].

3.2. SSH-Erweiterungen

OpenSSH implementiert zusätzlich zum SSH-Protokoll gemäss RFC 4250 bis 4254 („Core RFCs“) Erweiterungen in Form weiterer RFC-Dokumente („Extension RFCs“) [10]. Gegebenenfalls werden entsprechende Erweiterungen bereits in Kapitel 3.1 miteinbezogen, welche ebenfalls in folgendem Überblick aktuell implementierter Spezifikationen aufgeführt werden [4]:

- RFC 4255: Publizieren des SSH „Key-Fingerprints“ mittels DNS-Resource-Record des Typs „SSHFP“ (inkl. Absicherung mittels DNSSEC) [18]
  - RFC 6594: Zusätzliche Unterstützung von SHA-256 und ECDSA [38]
  - RFC 7479: Zusätzliche Unterstützung von Ed25519 [39]
  - RFC 8709: Zusätzliche Unterstützung von Ed448 [34]
- RFC 4256: Authentisierungsmethode `keyboard-interactive` zur Ermöglichung weiterer Authentisierungsmechanismen ohne spezifische Kenntnisse beim SSH-Client (z.B. „One Time Password“) [36]
- RFC 4335: Unterstützung des „BREAK“-Signals in einem Session-Kanal
  Dieses Signal wurde bei Verbindungen via Telnet oder seriellen Konsolen-Ports genutzt und konnte je nach Hersteller des Konsolen Servers administrative Funktionen ermöglichen [40]
- RFC 4344: Neue Transport Layer Verschlüsselungsmodi und Empfehlungen zum erneuten Schlüsselaustausch [22]
- RFC 4419: Neue Methoden für DHKE mit SHA-1 und SHA-256 [28]
- RFC 4462: Erweiterung im Transport Layer Protocol zum GSS-API (Generic Security Services Application Program Interface)-authentisierten DHKE und
  im Authentication Protocol zur Benutzerauthentisierung mittels GSS-API
  Die GSS-API kann z.B. mit Kerberos verwendet werden [41]
- RFC 4716: Beschrieb des Dateiformats von SSH-2-Public-Keys zum Austausch mittels Dateiübertragungsmechanismen [42]
- RFC 5656: Neue Algorithmen basierend auf der ECC (Elliptic Curve Cryptography) für das Transport Layer Protocol [29] sowie die Verwendung von SHA-2 [29]
- RFC 6668: Neue HMAC-SHA2-Algorithmen [23]
- RFC 8160: Erweiterung der Optionen für Terminal-Modi bei Sessions im Connection Protokoll (Encodierung „IUTF8“) [43]
- RFC 8270: Erhöhung der empfohlenen Minimalgrösse für Diffie-Hellman-Gruppen von 1024 auf 2048 Bits [25]
- RFC 8308: „Extension Negotiation“-Mechanismus zum Austausch weiterer Informationen während des Schlüsselaustauschs [35]
- RFC 8332: Erweiterung von RSA mit SHA-256 (`rsa-sha2-256`) und SHA-512 (`rsa-sha2-512`) [33]
- RFC 8709: Neue Public-Key-Algorithmen Ed25519 (`ssh-ed25519`) und Ed448 (`ssh-ed448`) [34]
- RFC 8731: Weitere ECDH-Methoden für Curve25519 und Curve448 [32]
- RFC 9142: Anpassung der empfohlenen Algorithmen zur Erhöhung der Sicherheit [21]

[4] Gemäss OpenSSH-Webseite [10] wurden zusätzliche Spezifikationen implementiert, die weder SSH-Protokoll Version 2 „Core RFCs“ noch „Extension RFCs“ sind [10]. Zusätzliche Erweiterungen durch OpenSSH selber sind in Kapitel 3.4 aufgeführt

3.3. SFTP

Das SFTP (SSH File Transfer Protocol) für sichere Dateiübertragungen über ein zuverlässiges Netzwerk ist in abgelaufenen RFC-Entwürfen („Drafts“) festgehalten [44].

Es wird über einen Secure Channel betrieben, der zum Beispiel in Form einer bereits authentisierten SSH-Verbindung auftreten kann [44]. Die Client-Benutzeridentität steht dem Protokoll dabei zur Verfügung, wobei der Client mittels SSH-Authentication-Protocol authentisiert wurde [44]. Bei der Verwendung mit SSH wird das Protokoll als Subsystem mit Namen `sftp` in einem SSH-Session-Channel gemäss Kapitel 3.1.5.1 eingebunden [44].

Dateipfade können absolut (beginnend mit „/“) oder relativ zum Standardverzeichnis des Benutzers angegeben werden [44]. Pfad-Komponenten wie `..` für Überverzeichnisse oder `.` für das derzeitige Verzeichnis sollten ebenfalls möglich sein [44]. Die SFTP-Server-Implementation ist für nötige Zugriffseinschränkungen zuständig, um z.B. Zugriffe für bestimmte Benutzer entsprechend einzuschränken [44].

Mittels SFTP können Dateien u. a. gelesen, gelöscht, geschrieben und umbenannt werden [44]. Verzeichnisse können u. a. angezeigt, erstellt und gelöscht werden [44]. Zusätzlich bestehen Funktionalitäten wie File-Hashing und Prüfung des verfügbaren Speicherplatzes [45].

3.4. OpenSSH

OpenSSH ist eine Software-Implementation des SSH-Protokolls und besteht aus folgenden Anwendungen, welche in den nächsten Kapiteln genauer erläutert werden [46]:

- Fernzugriffe bzw. Client-Software:
  - `ssh`: Remote Login Client [47]
  - `scp`: Secure File Copy [48]
  - `sftp`: Secure File Transfer (bietet gegenüber `scp` einen interaktiven Modus) [49]
- Schlüsselverwaltung
  - `ssh-add`: Hinzufügen von Identitäten mit dessen Private-Key zu `ssh-agent` [50]
  - `ssh-keysign`: Unterstützung der Host-basierten Authentisierung [51]
  - `ssh-keyscan`: SSH-Public-Keys von Server laden [52]
  - `ssh-keygen`: Schlüssel zur Authentisierung generieren, verwalten und konvertieren [53]
- Server-Software
  - `sshd`: SSH-Serverdienst (OpenSSH Daemon) [54]
  - `sftp-server`: SFTP Server Subsystem [55]
  - `ssh-agent`: Authentisierungs-Agent [56]

Gegebenenfalls wird zu einem späteren Zeitpunkt auf konkrete Einstellungsmöglichkeiten und Ressourcen eingegangen, welche innerhalb dieses Kapitels zur Wahrung der Übersichtlichkeit nicht aufgeführt werden.

OpenSSH hat zu den zusätzlich erwähnten Algorithmen eine eigene Auswahl an Algorithmen hinzugefügt, welche u. a. den Suffix „@openssh.com“ tragen (mehr zum Namensformat unter Kapitel 3.1.2) [10].

Wie bereits in Kapitel 1.2 erwähnt wird OpenSSH primär für das Betriebssystem OpenBSD entwickelt und für andere Produkte als „portable“ Version bereitgestellt, welche in ihrer Versionsbezeichnung mit einem „p“ markiert wird [4]. Diese Arbeit legt den Fokus auf die Implementation in Version 9.3 vom 19. Juli 2023 in OpenBSD 7.3 vom 10. April 2023.

3.4.1. Client-Software

3.4.1.1. SSH-Client ssh

`ssh` ist die SSH-Client-Software zum Einloggen und Ausführen von Befehlen auf entfernten Hosts, welche mit `sshd` auf SSH-Verbindungen von Clients warten [47][54].

Als Ziel-Parameter kann `[benutzer@]hostname` oder
eine URI in Form von `ssh://[user@]hostname[:port]` angegeben werden [47] [5].

Betreffend der Konfiguration von `ssh` werden die Optionen in folgender Priorität angewendet [57]:

1. Kommandozeilen-Parameter
2. Benutzerspezifische Konfigurationsdatei unter `~/.ssh/config` [6]
3. Systemweite Konfigurationsdatei unter `/etc/ssh/ssh_config`

Die Konfigurationsdateien ermöglichen es bestimmte Optionen für Hosts zu definieren, wessen Namen einem bestimmten Muster entsprechen müssen [57]. Mit `Host` und `Match` werden darauffolgende Parameter bis zum nächsten `Host` oder `Match` nur für zutreffende Hosts angewendet [57]. Muster und Konditionen für `Host` und `Match` können der `ssh_config`-Anleitung [57] entnommen werden.

[5] Angaben in eckigen Klammern („[“ „] “) sind optional

[6] „`~`“ entspricht dem „Home“-Verzeichnis des Benutzers, z.B. bei Benutzer `user` würde „`~`“ dem Pfad `/home/user` entsprechen

Eine Vielzahl von Parametern können `ssh` mitgegeben werden, wobei u. a. folgende Optionen zur Auswahl stehen (mit Angabe über die Kommandozeile oder der Konfigurationsdatei) [47][57]:

Beschreibung	Kommandozeile	Konfigurationsdatei
Aktiviert/Deaktiviert die Verbindungsweiterleitung eines **Authentisierungs-Agenten** wie `ssh-agent`	`-A` / `-a`	`ForwardAgent` `yes` / `no`
Angabe der **Verschlüsselungsalgorithmen** als Komma-separierte Liste	`-c`	`Ciphers`
Pfad zur alternativen, benutzerspezifischen **Konfigurationsdatei** Die systemweite Konfigurationsdatei wird dann ignoriert	`-F configfile`	-
Ausgabe der effektiven Konfiguration nach Evaluation von `Host` und `Match` für das angegebene Ziel	`-G`	-
Angabe der Identität für die **Public-Key-Authentisierung** in Form einer Private-Key-Datei Mehrere Keys sind durch weitere Angaben möglich Angabe einer Public-Key ist auch eine Option, sofern der zugehörige Private-Key mit `ssh-agent` geladen ist	`-i`	`IdentityFile`
Verwenden einer oder mehrerer **Jump Hosts** (TCP-Forwarding zum Ziel)	`-J`	`ProxyJump`
Anbindung einer lokalen Adresse, einem Port oder Unix Socket **zu einem entfernten Zielpunkt**, sodass bei einer Verbindung zum lokalen Punkt diese weitergeleitet wird (**„Local Port Forwarding“**)	`-L`	-
Aktivierung/Deaktivierung der Nutzung von **„Local Port Forwardings“**, sodass sie **von anderen Clients genutzt** werden können Anbindung auf eine lokale (Deaktivierung) oder von aussen zugängliche Adresse (Aktivierung)		`GatewayPorts` `yes` / `no`
MAC-Algorithmen als Komma-separierte Liste	`-m`	`MACs`
Konfiguration gemäss Optionsformat der Konfigurationsdatei u. a. für **Optionen ohne Kommandozeilenparameter** Gültige Schreibweisen: `-o VerifyHostKeyDNS=yes` und `-o "VerifyHostKeyDNS yes"`	`-o`	-
Port des Ziel-Hosts	`-p`	`Port`
Ausgabe unterstützter Algorithmen wie z.B. `cipher` oder `mac`	`-Q`	-
Wahl des **Session-Typs** (Befehl, Subsystem oder „none“ für reines Port-Forwarding)	`-s`	`SessionType`
Ausgabe von **Debugging Meldungen** in 3 Detailgraden	`-v` / `-vv` / `-vvv`	-
Aufbau eines **Tunnels** (z.B. VPN) zur Verbindung zweier Netzwerke	`-w`	`Tunnel` / `TunnelDevice`

Tabelle 3.1.: Auszug SSH-Client-Konfigurationsparameter [47][57]

Die zuvor aufgelistete Auswahl weist die Flexibilität des SSH-Clients auf, wobei weitere Optionen u. a. zur Kompression, ein Batch-Mode für die Verwendung in Skripts, Normalisierung der Hostnamen („canonicalization"), Wahl der Algorithmen für Schlüsselaustausch, Public-Key-Authentisierung, sowie Zertifikaten verfügbar sind [57]. Eine komplette Auflistung der `ssh`-Parameter kann dessen Manpage [47] sowie der Manpage zur zugehörigen Konfigurationsdatei [57] entnommen werden.

`ssh` unterstützt die in Kapitel 3.1.4 und 3.2 aufgelisteten Authentisierungsmechanismen, wessen dafür konkret betrachtete Ressourcen in der `ssh`-Manpage [47] aufgeführt und ggf. zu einem späteren Zeitpunkt aufgegriffen werden [47]. Hosts, mit welchen `ssh` kommuniziert hat, werden mit ihren Public-Keys unter `~/.ssh/known_hosts` abgelegt [47]. Somit kann bei einem Wechsel eines Public-Keys bei einem bekannten Host eine Warnung ausgegeben werden, um Man-in-the-Middle-Angriffe vorzubeugen [47].

Während einer Terminal-Verbindung mit `ssh`, können mittels eines „Escape-Charakters"[7] Funktionen wie z.B. das Anzeigen weitergeleiteter Verbindungen (`~#`), ein Anfordern eines erneuten Schlüsselaustauschs (`~R`) oder das Anzeigen unterstützter Escape-Sequenzen (`~?`) angewendet werden.

```
c1$ ssh user@192.168.1.1
The authenticity of host '192.168.1.1 (192.168.1.1)' can't be established.
ED25519 key fingerprint is SHA256:ctWpch78E8V8qZEv74v72HQaHVTauCgB0xF5xF1OvRU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.1.1' (ED25519) to the list of known hosts.
user@192.168.1.1's password:
Last login: Wed Aug  9 16:37:21 2023 from 192.168.100.1
OpenBSD 7.3 (GENERIC) #3: Tue Jul 25 08:18:48 MDT 2023

Welcome to OpenBSD: The proactively secure Unix-like operating system.

...

s1$ ~?
Supported escape sequences:
~.   - terminate connection (and any multiplexed sessions)
~B   - send a BREAK to the remote system
~R   - request rekey
~V/v - decrease/increase verbosity (LogLevel)
~^Z  - suspend ssh
~#   - list forwarded connections
~&   - background ssh (when waiting for connections to terminate)
~?   - this message
~~   - send the escape character by typing it twice
(Note that escapes are only recognized immediately after newline.)
```

Quelltext 3.1: SSH-Client Verbindungsaufbau zu einem neuen Server und Anzeige „Escape Sequenzen"

[7] Der Standard-Escape-Charakter ist „`~`", kann aber mittels Option `-e` oder `EscapeChar` angepasst werden [57]. Für ein schweizerisches Tastaturlayout wäre evtl. „`?`" geeigneter.

3.4.1.2. Secure File Copy scp

`scp` ermöglicht das Kopieren von Dateien zwischen Hosts in einem Netzwerk mittels SFTP (SSH File Transfer Protocol) [48]. Die Applikation macht Gebrauch einer Verbindung mittels `ssh`, verwendet somit deren Konfigurationsdateien und verfügt über ähnliche Parameter, wobei es auch welche mit selber Kommandozeilenoption, aber anderem Nutzen gibt [48][47]. Zum Beispiel ist für die Angabe des Ziel-Ports `-P` anstelle von `-p` zu wählen (bei letzterem werden Datei-Metadaten gewahrt bzw. „preserved“) [48]. Die Verwendung u. a. eines Authentisierungs-Agenten, alternativen Konfigurationsdateien oder Jump Hosts ist wie bei `ssh` ebenfalls möglich [48].

Es sind Quell- und Zielpfade zu verwenden, welche lokale und entfernte Pfade sein können [48]. Entfernte Pfade können, ähnlich wie bei `ssh`, in der Form von `[benutzer@]hostname:[pfad]` oder in URI-Form von `scp://[user@]hostname[:port][/pfad]` angegeben werden [48]. Absolute und relative Pfade (je nachdem relativ zum Benutzer-Heimverzeichnis oder derzeitigen Pfad) sind möglich [48].

Seit OpenSSH Version 9.0 vom 8. April 2022 [58] verwendet `scp` das Protokoll SFTP anstelle von SCP (Secure Copy Protocol), wobei SCP mit der Option `-O` noch für ältere Zielsysteme verwendet werden kann [48]. Die aktuell von OpenSSH verwendete SFTP-Protokoll-Version ist 3, welche im entsprechenden RFC-Draft Version 2 festgehalten ist, wobei dieser Draft mit SFTP-Protokoll-Version 6 bereits in Version 13 festgehalten wurde [10][59].

Dadurch, dass `scp` eine Verbindung über `ssh` aufbaut, können mittels `-o` Optionen für `ssh` mitgegeben werden [48].

```
c1$ scp document.pdf user@192.168.1.1:
user@192.168.1.1's password:
document.pdf                  100%  847KB  13.2MB/s   00:00
```

Quelltext 3.2: Kopieren einer Datei zu einem Server mit `scp`

3.4.1.3. Secure File Transfer sftp

Mittels `sftp` werden Dateiübertragungen via SFTP mit Hilfe von `ssh` durchgeführt, wodurch es ähnlich wie `scp` verwendet werden kann [49]. Die Kommandozeilenparameter ähneln somit ebenfalls `scp` und `ssh`, wobei diese näher bei ersterer Anwendung sind.

Im Vergleich zu `scp` bietet `sftp` einen interaktiven Modus, welcher FTP-ähnliche Befehle verwendet wie z.B. `cd` für einen Verzeichniswechsel, `get` zum Datei-Download oder `put` zum Datei-Upload [49]. Somit können zusätzlich zum Kopieren von Dateien mit `sftp` weitere Operationen wie das Umbenennen oder Löschen von Dateien getätigt werden [60].

Es ist ein entfernter Zielpfad anzugeben, welcher wie bei `scp`, in der Form von `[benutzer@]hostname:[pfad]` oder in URI-Form von `scp://[user@]hostname[:port][/pfad]` auftreten kann [49]. Bei der Angabe eines Ordners oder Weglassen des Pfades wird der interaktive Modus gestartet [49]. Handelt es sich beim Zielpfad um eine Datei, wird diese nach der Authentifizierung (welche nicht interaktiv sein kann) automatisch heruntergeladen [49].

```
c1$ sftp user@192.168.1.1
user@192.168.1.1's password:
Connected to 192.168.1.1.
sftp> ls
document.pdf
sftp> get document.pdf /tmp/documentdownloaded.pdf
Fetching /home/user/document.pdf to /tmp/documentdownloaded.pdf
document.pdf                  100%  847KB  13.6MB/s   00:00
sftp> exit
```

Quelltext 3.3: Download einer Datei von einem Server mit `sftp` im interaktiven Modus

3.4.2. Key-Management-Software

3.4.2.1. Authentisierungs-Agent-Identitätenverwaltung mit ssh-add

Mit `ssh-add` können beim Authentifizierungs-Agenten `ssh-agent` Private-Key-Identitäten gegebenenfalls mit zugehörigem Zertifikat hinzugefügt oder entfernt werden [50]. Zur erfolgreichen Anwendung muss `ssh-agent` bereits gestartet sein [50].

Schlüssel können für eine mitgegebene Zeitdauer und/oder mit Einschränkung zur Nutzung für bestimmte Benutzer und Ziele hinzugefügt werden, wobei bei einer Weiterleitung sämtliche Hosts in der Kette erlaubt sein müssen [50].

`ssh-agent` kann mit einem eigenen Passwort mittels `ssh-add -x` gesperrt und mit `ssh-add -X` entsperrt werden [8] [50]. Eine Unterstützung für PKCS#11 und FIDO Authenticators ist ebenfalls vorhanden, um z.B. Keys von einem kryptografischen Token verwenden zu können [50].

Folgender Auszug an Optionen zur Handhabung von `ssh-agent` zeigt verschiedenste Möglichkeiten mit `ssh-add` auf [50]:

Beschreibung	Kommandozeile
Hinzugefügte Identität ist **vor** dessen **Verwendung zu bestätigen**	`-c`
Entfernt sämtliche Identitäten von `ssh-agent`	`-D`
Verknüpft die hinzugefügte Identität mit der **Bedingung**, nur auf bestimmten Zwischen- und/oder Ziel-Hosts zu verwenden	`-h bedingung`
Auflistung sämtlicher **Public-Key-Parameter** der geladenen Identitäten	`-L`
Auflistung sämtlicher **Fingerprints** der geladenen Identitäten	`-l`
Definition einer **maximalen Lebensdauer** in Sekunden oder gemäss Format aus Manpage von `sshd_config` [3] für die hinzugefügte Identität	`-t lebensdauer`
Ausgabe von **Debugging Meldungen** in 3 Detailgraden	`-v` / `-vv` / `-vvv`
Entsperren von `ssh-agent`	`-X`
Sperren von `ssh-agent` mit einem Passwort	`-x`

Tabelle 3.2.: Auszug Parameter zur Authentisierungs-Agenten-Handhabung mit `ssh-add` [50]

3.4.2.2. Host-basierte Authentisierung mit ssh-keysign

`ssh-keysign` wird von `ssh` für die Signatur-Generierung lokaler Private-Keys zur Host-basierten Authentisierung verwendet [51]. Die Funktion zur Verwendung dieser Applikation (`EnableSSHKeysign`) ist im Auslieferungszustand deaktiviert [51].

[8] mit einem gesperrten `ssh-agent` können dessen geladene Keys nicht verwendet werden

3.4.2.3. SSH-Public-Keys laden mit ssh-keyscan

Mit `ssh-keyscan` werden öffentliche SSH-Schlüssel von Hosts abgefragt, welche mittels Hostnamen, IP-Adresse oder Netzwerk-Adressbereich angegeben werden können [52].

Die Anwendung dient zur Unterstützung beim Aufbau und der Verifikation der lokalen Identifikationsdatenbank des SSH-Clients `ssh` (`~/.ssh/known_hosts`) [52]. Analog zu `ssh` sind die Schlüssel selbst zu verifizieren, um nicht verwundbar gegen Man-in-the-Middle-Angriffe zu sein (Wiederholte Scans können jedoch geänderte Schlüssel aufzeigen) [52].

```
c1$ ssh-keyscan -t rsa,dsa,ecdsa,ed25519 192.168.1.1
# 192.168.1.1:22 SSH-2.0-OpenSSH_9.3
# 192.168.1.1:22 SSH-2.0-OpenSSH_9.3
192.168.1.1 ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDOEPLVFiiOS3S+mv1rOUk+FIv0M
   +7rR/Of+UjRDeB7fBieX2U7au3bniXG2JwkJGI6Rv7eb+8B8nM+vaUIQvcQzIjBUR8/fSyCC1
   rTU85na7TnNZ1akDL170lxCk1ERvz+iGM/rzNk8ptm8p8xpxjDuC10BAxDicEyI0yZAWHGz9
   sMlZBINHgOrgwy5wv4pBjW2v57e6sGFKp4Nd8BpbipLKXudgte3hBY7WBLpXJC3zV/GEBL5
   FjAdaAEnm0Fyr5rH9mHpNE9QRSj3qKHTtan2TlIOFyGMkzn31H4lpSdweD38yFtVXVle2/+L5
   QNReA5wFzB4im9Yh8KEYQ1c4CL5gIwxBnNNytwIYTbsfDDqCkwsdDtRIVFSDiI7F9omaJKD1zN
   5lmoaQsmuETiJjjzaXn2g2HZuZCPMngL13KHmMMVcCLTzDSMKN1efIqjXfSJCVaPMXyxPdYra6
   g7fAIOJdYfDufvIAsUejEsVsD1FT1TCu1tbQkGR8XQAzK7OD8=
# 192.168.1.1:22 SSH-2.0-OpenSSH_9.3
192.168.1.1 ecdsa-sha2-nistp256 AAAAE2
   VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBAdhVOH/r6DjXA8pOVP32
   seJkClQkiJnrG4Oxc3E4cSzmp7m3XnAA7qEZMi0ZmaXZyG8SMy1WSOw3KjU0QVcUdw=
# 192.168.1.1:22 SSH-2.0-OpenSSH_9.3
192.168.1.1 ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIGBtIS+9ThfcrThgvy3HYcz3ukuR6
   rXo0Z7mHwLN5P1D
```

Quelltext 3.4: Mit `ssh-keyscan` Public-Keys bestimmter Typen laden

3.4.2.4. Schlüsselgenerierung und -Handhabung mit ssh-keygen

Schlüssel zur Authentisierung mit `ssh` können mit `ssh-keygen` generiert, verwaltet und konvertiert werden (Benutzerspezifische Schlüssel für die Public-Key-Authentisierung sowie Host-Schlüssel für die SSH-Server) [53]. Zusätzlich kommt die Anwendung bei den Schlüsselaustauschmethoden `diffie-hellman-group-exchange-sha1` und `diffie-hellman-group-exchange-sha256` zum Zug, zu welchen entsprechende Diffie-Hellman Gruppen generiert werden [53].

„Key Revocation Lists“, in welchen für ungültig erklärte Schlüssel festgehalten werden, können mit `ssh-keygen` ebenfalls erstellt und aktualisiert werden [53].

Erstellte Schlüssel werden je nach Benutzerangabe und/oder gewähltem Algorithmus bzw. Schlüssel-Typen wie z.B. RSA, ECDSA oder Ed25519 unter einem bestimmten Pfad gespeichert, wobei der zugehörige Public-Key mit dem Suffix „.pub“ abgelegt wird [53]. Auf Wunsch kann dem Schlüssel ein Passwort mitgegeben werden, was den Private-Key mit 128-Bit-AES verschlüsselt, bei dessen Verwendung angegeben werden muss und später geändert werden kann [53].

Das Format der Schlüssel entspricht einem OpenSSH-eigenen Format, jedoch können mit dem Parameter `-m` andere Formate wie `RFC4716` [42], `PEM` [61] oder `PKCS8` [62] verwendet werden [53].

Ein Kommentar zur Unterstützung der Schlüssel-Identifikation wird beim Erstellen mit `user@host` erstellt, kann aber mit der Option `-c` angepasst werden [53].

Die Schlüsseltypen und -Grössen können (sofern möglich) gewählt werden, z.B. `-t rsa` und `-b 3072` für RSA mit 3072 Bit [53].

Bestehende Schlüssel können eingelesen werden, um deren Schlüssel-Formate (wie z.B. `RFC4716`) umzuwandeln, Public-Keys oder Werte für u. a. DNS-SSHFP-Records zu exportieren [53].

OpenSSH-Zertifikate (für Benutzer oder Hosts) können mittels Schlüssel-Signierung und ggf. einem CA-Schlüssel und/oder einer bestimmten Lebensdauer erstellt und validiert werden [53].

```
c1$ ssh-keygen -t ed25519 -f /tmp/test_ed25519_key -C "Comment for test key"
Generating public/private ed25519 key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /tmp/test_ed25519_key
Your public key has been saved in /tmp/test_ed25519_key.pub
The key fingerprint is:
SHA256:p5tWd93keRT+IwjpLFgZECq4BZH39OH531Md+S/EHWg Comment for test key
The key's randomart image is:
+--[ED25519 256]--+
|oo   oo          |
|o.. o ..       . |
|.o.+ o oo .   o..|
| o. . +o o   Eooo|
|.     o.S o oo O=|
|     . ..=..o.=oO|
|       .oo o o. +|
|        ooo   . .|
|       .o  .   . |
+----[SHA256]-----+
c1$ cat /tmp/test_ed25519_key.pub
ssh-ed25519
   AAAAC3NzaC1lZDI1NTE5AAAAIJNAHFjL48jRyCdhPZWIKa3yQfb5cBJ0MJF2C2v34iGD
   Comment for test key
c1$ rm /tmp/test_ed25519_key*
```

Quelltext 3.5: Generierung von Ed25519 Private- und Public-Keys

3.4.3. Server-Software

3.4.3.1. SSH-Server sshd

Der SSH-Serverdienst von OpenSSH trägt den Namen `sshd` (OpenSSH Daemon) und hört auf Verbindungen von SSH-Clients [54]. Analog zu besagtem Client aus Kapitel 3.4.1.1 können Konfigurationen mittels Kommandozeilenparameter oder der Konfigurationsdatei `/etc/ssh/sshd_config` mitgegeben werden, wobei die Einstellungen durch die erste Option Vorrang haben [54]. Für jede neue eingehende Verbindung wird der Daemon zu einem neuen Prozess dupliziert („Fork“), welcher die gesamte Verbindung inklusive Schlüsselaustausch, Verschlüsselung, Authentisierung, etc. handhabt. [54].

Folgender Auszug an Kommandozeilenparametern für `sshd` zeigt damit mögliche Vorgänge auf [9] [54]:

Beschreibung	Kommandozeile
Anwendung **nicht im Hintergrund** ausführen Deaktivierung des Daemon-Verhaltens	`-D`
Ausgabe von **Debugging Meldungen** in 3 Detailgraden	`-d` / `-dd` / `-ddd`
Pfad zu einer alternativen **Konfigurationsdatei**	`-f configfile`
Ausgabe der effektiven Konfiguration	`-G`
Definition eines **Zeitlimits zur Authentisierung** Standard beträgt 120 Sekunden	`-g login_grace_time`
Angabe eines oder mehreren **spezifischen Host-Keys** Je nach Algorithmus	`-h host_key`
Konfiguration gemäss Optionsformat der Konfigurationsdatei u. a. für **Optionen ohne Kommandozeilenparameter**	`-o`
Port, auf welchem der Dienst aktiv sein soll Standard ist Port 22 Mittels `ListenAddress` in der Konfigurationsdatei definierte Ports haben Vorrang	`-p`
Test Modus, in welchem die Gültigkeit der Konfigurationsdatei und die Verwendbarkeit der Schlüssel geprüft wird	`-t`
Erweiterter Test Modus, welcher ebenfalls die Konfigurations-Gültigkeit prüft und die effektive Konfiguration ausgibt, ggf. nach Evaluation von `Match`-Parametern mittels `-C` `Match`-Parameter (`-C`) Beispiel: `-C "user=cmd"` Die Option ist somit eine Kombination von `-G` und `-t`	`-T`

Tabelle 3.3.: Auszug SSH-Server-Konfigurationsparameter (Kommandozeile) [54]

[9] Parameter der Konfigurationsdatei `sshd_config` werden in einem späteren Abschnitt aufgezeigt

Gemäss Manpage von `sshd` verläuft die Client-Server-Kommunikation nach folgendem Ablauf [54]:

1. Verbindung und Schlüsselaustausch
   a) Der **Client verbindet** zum Server und erhält von diesem dessen öffentlichen Host-Schlüssel
   b) Der **erhaltene Server-Schlüssel wird vom Client geprüft**, ob ihm dieser bekannt ist oder geändert hat
   c) Der **Diffie–Hellman Key Exchange** (Schlüsselaustausch) wird durchgeführt, woraus ein **gemeinsamer Session-Key** resultiert
   d) Die darauf folgende **Übertragung** wird **symmetrisch mit dem Session-Key verschlüsselt**
      Der Client wählt den zu verwendenden Algorithmus aus der vom Server offerierten Liste
   e) Zur Prüfung der Sitzungsintegrität wird ein **MAC (Message Authentication Code)** verwendet
2. Server und Client gehen einen **Authentisierungs-Dialog** ein, wobei sich der Client mit entsprechenden Methoden wie Public-Key- oder Passwort-Authentisierung zu authentisieren versucht
   Unter der Option der `sshd`-Konfiguration mit Namen `AuthorizedKeysFile` wird die Datei referenziert, welche erlaubte Public-Keys für die Public-Key-Authentisierung enthält. Die Angabe mehrerer Dateien ist möglich, welche zusätzlich Schlüsselspezifische Optionen beinhalten können, die u. a. bestimmte Funktionen oder Weiterleitungen aktivieren/deaktivieren können sowie die Forcierung eines Ablaufdatums oder spezifischen Befehls erlauben.
   Der Standardwert der `AuthorizedKeysFile`-Option lautet `.ssh/authorized_keys .ssh/authorized_keys2`
3. Bei erfolgreicher Authentisierung wird die SSH-Session vorbereitet
   Unter anderem das Allozieren eines Pseudo-Terminals sowie Weiterleitungen von X11- oder TCP-Verbindungen können vom Client beantragt werden
4. Nach **Aufbau der Session** fragt der Client **eine interaktive Shell oder einen bestimmten Befehl** an, worauf beide in den „Session Mode“ gehen In diesem Modus können beide Seiten Daten senden
5. `sshd` operiert **nach einem erfolgreichen Benutzerlogin** wie folgt:
   a) Bei einem Login auf einem Terminal ohne spezifischen Befehl zeigt der Server gegebenenfalls den **Zeitstempel des letzten Logins** sowie den Inhalt von `/etc/motd` („Message of the day “) an und zeichnet den aktuellen Login-Zeitstempel auf
   b) **Existiert die Datei `/etc/nologin` zur Verweigerung von Logins**, wird dessen Inhalt ausgegeben und die Verbindung beendet
      Ausnahme bildet der User `root`, bei dem die Sitzung weiterläuft
   c) Der Prozess wechselt in die **Ausführung mit normalen Benutzerprivilegien** und baut eine einfache Umgebung auf
   d) Sollte das Laden von benutzerspezifischen **Umgebungsvariablen** gemäss `sshd`-Konfiguration mit `PermitUserEnvironment` gestattet sein, so werden diese aus der Datei `~/.ssh/environment` oder der Option `environment=` aus dem `AuthorizedKeysFile` gelesen
   e) Der `sshd`-Prozess wechselt in das **Heimverzeichnis des Benutzers**
   f) **Je nach Konfiguration** werden, sofern diese Dateien existieren, sogenannte **„RC-Skripts“** ausgeführt, die entsprechende Initialisierungsroutinen beinhalten können [10]
   g) Der vom Benutzer angegebene **Befehl oder hinterlegte Shell-Prozess wird ausgeführt**
      Befehle werden unter der hinterlegten Shell ausgeführt

[10] Bei aktivierter Option `PermitUserRC` (Standardmässig aktiviert) wird ggf. das RC-Skript unter `~/.ssh/rc` ausgeführt, bei deaktivierter Option ggf. `/etc/ssh/sshrc` [54]. Beide Dateien existieren in einer neuen OpenBSD-Installation nicht

6. Bei Beendung der Benutzerapplikation (kann auch eine Shell sein) und Schliessung sämtlicher weitergeleiteter Verbindungen sendet der Server den „Exit Status“ zum Client, worauf beide den **Prozess beenden**

Aus der Perspektive des SSH-Clients sieht dieser Ablauf wie folgt aus:

```
c1$ ssh -v user@192.168.1.1
OpenSSH_9.3, LibreSSL 3.7.2
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Connecting to 192.168.1.1 [192.168.1.1] port 22.
...
debug1: Local version string SSH-2.0-OpenSSH_9.3
debug1: Remote protocol version 2.0, remote software version OpenSSH_9.3
...
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: algorithm: sntrup761x25519-sha512@openssh.com
debug1: kex: host key algorithm: ssh-ed25519
debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC: <
    implicit> compression: none
debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC: <
    implicit> compression: none
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: SSH2_MSG_KEX_ECDH_REPLY received
debug1: Server host key: ssh-ed25519 SHA256:ctWpch...1OvRU
...
debug1: Host '192.168.1.1' is known and matches the ED25519 host key.
debug1: Found key in /home/user/.ssh/known_hosts:1
debug1: rekey out after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: rekey in after 134217728 blocks
debug1: Will attempt key: /home/user/.ssh/id_...
debug1: SSH2_MSG_EXT_INFO received
debug1: kex_input_ext_info: server-sig-algs=<ssh-ed25519,...
debug1: kex_input_ext_info: publickey-hostbound@openssh.com=<0>
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password,keyboard-
    interactive
debug1: Next authentication method: publickey
debug1: Trying private key: /home/user/.ssh/id_...
debug1: Next authentication method: keyboard-interactive
debug1: Authentications that can continue: publickey,password,keyboard-
    interactive
debug1: Next authentication method: password
user@192.168.1.1's password:
Authenticated to 192.168.1.1 ([192.168.1.1]:22) using "password".
debug1: channel 0: new session [client-session] (inactive timeout: 0)
...
Last login: Wed Aug  9 17:43:29 2023 from 192.168.100.1
OpenBSD 7.3 (GENERIC) #3: Tue Jul 25 08:18:48 MDT 2023
s1$ exit
...
Connection to 192.168.1.1 closed.
```

Quelltext 3.6: Client-Server-Kommunikation gemäss SSH-Client-Debugging-Meldungen

SSH-Server Konfigurationsdatei sshd_config

Die Konfigurationsdatei für `sshd` ist standardmässig unter `/etc/ssh/sshd_config` vorzufinden und bietet über 90 Optionen [54][3]. Einzelne davon unterstützen die Verwendung von Schlüsselwörtern, welche zur Laufzeit ausgewertet werden und zum Beispiel dem Heimverzeichnis des Benutzers, einem Schlüssel-Typen, -Fingerprint oder dem Benutzernamen entsprechen können [3].

Da im Rahmen dieser Arbeit primär der SSH-Serverdienst behandelt wird, werden sämtliche Optionen der zugehörigen Manpage [3] in eigens definierte Kategorien aufgeteilt und beschrieben.

Algorithmen-Wahl [3]

- `CASignatureAlgorithms`
  Definiert die erlaubten **Algorithmen**, mit welchen von **CA signierte Zertifikate** geprüft werden
  Zertifikate mit nicht hinterlegten Algorithmen werden bei der **Host-basierten- oder Public-Key-Authentisierung** abgelehnt
  Standardeinstellung: `ssh-ed25519, ecdsa-sha2-nistp256,ecdsa-sha2-nistp384, ecdsa-sha2-nistp521, sk-ssh-ed25519@openssh.com, sk-ecdsa-sha2-nistp256@openssh.com, rsa-sha2-512,rsa-sha2-256`
- `Ciphers`
  Spezifiziert die erlaubten **Algorithmen zur symmetrischen Verschlüsselung** (siehe Liste in Kapitel 3.4.3.1, Punkt 1d)
  Standardeinstellung: `chacha20-poly1305@openssh.com, aes128-ctr,aes192-ctr,aes256-ctr, aes128-gcm@openssh.com,aes256-gcm@openssh.com`
- `FingerprintHash`
  Legt den verwendeten Hash-Algorithmus für geloggte **Fingerprints** fest
  Standardeinstellung: `sha256`
- `HostbasedAcceptedAlgorithms`
  siehe Kategorie **„Authentisierung und Autorisierung: Host-basierte Authentisierung“**
- `HostKeyAlgorithms`
  siehe Kategorie **„Server-Schlüssel und -Zertifikate“**
- `KexAlgorithms`
  Beschreibt die verfügbaren **Algorithmen für den Schlüsselaustausch** (**KEX (Key Exchange)**, siehe Kapitel 3.1.3.3 und 3.1.3.5)
  Standardeinstellung [11]: `sntrup761x25519-sha512@openssh.com, curve25519-sha256, curve25519-sha256@libssh.org, ecdh-sha2-nistp256, ecdh-sha2-nistp384,ecdh-sha2-nistp521, diffie-hellman-group-exchange-sha256, diffie-hellman-group16-sha512 diffie-hellman-group18-sha512, diffie-hellman-group14-sha256`
- `MACs`
  Listet die verfügbaren **MAC-Algorithmen** (siehe Kapitel 3.1.3.2) auf
  Standardeinstellung [12]: `umac-64-etm@openssh.com, umac-128-etm@openssh.com, hmac-sha2-256-etm@openssh.com, hmac-sha2-512-etm@openssh.com, hmac-sha1-etm@openssh.com, umac-64@openssh.com, umac-128@openssh.com,hmac-sha2-256, hmac-sha2-512,hmac-sha1`
- `ModuliFile`
  Verweist auf eine Datei, welche für die DHKE-Schlüsselaustausch-Algorithmen `diffie-hellman-group-exchange-sha1` / `-sha256` zu verwendende **Diffie-Hellman-Gruppen** beinhalten
  Standardeinstellung: `/etc/moduli`
- `PubkeyAcceptedAlgorithms`
  siehe Kategorie **„Authentisierung und Autorisierung: Public-Key-Authentisierung“**

[11] Seit OpenSSH 9.0 vom 8. April 2022 wird mit der Schlüsselaustauschmethode `sntrup761x25519-sha512@openssh.com` („Streamlined NTRU Prime “ mit X25519 key exchange) als präferierter Algorithmus gesetzt, welcher theoretisch resistent gegen Angriffe von Quantencomputern ist [63][64]

[12] MAC-Algorithmen mit dem Suffix `-etm` berechnen den MAC nach der Verschlüsselung („Encrypt-then-MAC“) [3]

Authentisierung und Autorisierung: Allgemein [3]

- `AuthenticationMethods`
  Spezifiziert die durchzuführenden **Authentisierungsmethoden** gemäss Kapitel 3.1.4, wobei mit dem Beispiel-Parameter `publickey,password publickey,keyboard-interactive` eine Public-Key-Authentisierung zusammen mit einer Passwort- oder „Keyboard-Interactive“-Authentisierung durchgeführt werden muss
  Standardeinstellung: `any` (erlaubt jede Authentisierungsmethode)
- `GSSAPIAuthentication`
  Aktiviert/Deaktiviert die **Benutzerauthentisierung mittels GSS-API**
  Standardeinstellung: `no`
- `GSSAPICleanupCredentials`
  Aktiviert/Deaktiviert die Funktionalität, welche bei einer GSS-API-Authentisierung die zwischengespeicherten Benutzerdaten („user’s credentials cache“) beim Ausloggen zerstört
  Standardeinstellung: `yes`
- `GSSAPIStrictAcceptorCheck`
  Legt fest, ob bei der GSS-API-Authentisierung ein spezifischer Service mit dem derzeitigen Computernamen (`yes`) oder einen beliebigen Service aus dessen Speicher (`no`) anfragen darf
  Dieser Parameter dient zur Unterstützung von Geräten, die an mehreren Netzwerken gleichzeitig angebunden sind (sogenannte „Multi Homed Machines “)
  Standardeinstellung: `yes`
- `KbdInteractiveAuthentication`
  Aktiviert/Deaktiviert die **„Keyboard-Interactive“-Authentisierung** (siehe Kapitel 3.1.4.4), wobei die Login-Klassen gemäss Manpage von `login.conf` [65] wie zum Beispiel `passwd` (lokale Passwort-Datenbank) oder `ldap` (Verzeichnisprotokoll für u. a. eine zentrale Benutzerverwaltung) unterstützt werden
  Standardeinstellung: `yes`
- `KerberosAuthentication`
  Aktiviert/Deaktiviert bei der Passwort-Authentisierung (siehe Kapitel 3.1.4.2, festgelegt durch `PasswordAuthentication`), ob das **Passwort über Kerberos** mit dessen Kerberos KDC **validiert** werden soll
  Standardeinstellung: `no` (Deaktiviert)
- `KerberosGetAFSToken`
  Spezifiziert, ob bei aktiviertem AFS (verteiltes Netzwerk-Filesystem) [66] und vorhandenem Kerberos-Ticket (erhalten nach erfolgreicher Authentisierung via Kerberos [67]) zusätzlich ein AFS-Token (spezielles Kerberos-Ticket [66] angefordert werden soll)
  Standardeinstellung: `no`
- `KerberosOrLocalPasswd`
  Legt fest, ob bei **fehlgeschlagener Authentisierung mittels Kerberos** die **Passwort**-Validierung mit einem **lokalen** Mechanismus **geprüft** werden soll
  Standardeinstellung: `yes`
- `KerberosTicketCleanup`
  Aktiviert/Deaktiviert die Funktionalität, welche bei einer Kerberos-Authentisierung die **zwischengespeicherten Benutzerdaten** („user’s ticket cache“) beim Ausloggen **zerstört**
  Standardeinstellung: `yes`
- `PasswordAuthentication`
  Aktiviert/Deaktiviert die **Passwort-Authentisierung** (siehe Kapitel 3.1.4.2)
  Standardeinstellung: `yes` (Aktiviert)
- `PermitEmptyPasswords`
  Definiert, ob bei aktivierter Passwort-Authentisierung **leere Passwörter** verwendet werden dürfen
  Standardeinstellung: `no`
- `RequiredRSASize`
  Legt die Minimal-**Schlüsselgrösse für RSA-Schlüssel** in Bits fest
  Kleinere Schlüssel werden bei der **Public-Key- und Host-basierten-Authentisierung** abgelehnt
  Standardeinstellung: `1024`

Authentisierung und Autorisierung: Host-basierte Authentisierung (siehe Kapitel 3.1.4.3) [3]

- `HostbasedAcceptedAlgorithms`
  Definiert die **erlaubten Algorithmen** für Schlüssel **zur Host-basierten Authentisierung**
  Standardeinstellung: `ssh-ed25519-cert-v01@openssh.com,ecdsa-sha2-nistp256-cert-v01@openssh.com, ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com, sk-ssh-ed25519-cert-v01@openssh.com,sk-ecdsa-sha2-nistp256-cert-v01@openssh.com, rsa-sha2-512-cert-v01@openssh.com,rsa-sha2-256-cert-v01@openssh.com, ssh-ed25519,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521, sk-ssh-ed25519@openssh.com,sk-ecdsa-sha2-nistp256@openssh.com,rsa-sha2-512,rsa-sha2-256`
- `HostbasedAuthentication`
  Aktivierung/Deaktivierung der **Host-basierten Authentisierung**
  Standardeinstellung: `no` (Deaktiviert)
- `HostbasedUsesNameFromPacketOnly`
  Spezifiziert, ob der Server den **Client-Hostnamen zur Evaluation zuerst mittels DNS rückwärts auflöst** (`no`) oder den vom Client mitgeteilten Namen verwendet (`yes`)
  Standardeinstellung: `no`
- `IgnoreRhosts`
  Legt fest, ob **Benutzer-spezifische Dateien** zur Host-basierten Authentisierung (`.rhosts`/`.shosts`) betrachtet werden sollen (`no`) oder ob nur systemweite Dateien (`/etc/hosts.equiv`/`/etc/shosts.equiv`) verwendet werden sollen (`yes`)
  Standardeinstellung: `yes`
- `IgnoreUserKnownHosts`
  Besagt, ob bei der Host-basierten Authentisierung die **Benutzer-spezifische Liste bekannter Host-Schlüssel** (`~/.ssh/known_hosts`) oder nur die systemweite Liste (`/etc/ssh/ssh_known_hosts`) verwendet werden soll
  Standardeinstellung: `no` (beide Listen beachten)

Authentisierung und Autorisierung: Public-Key-Authentisierung (siehe Kapitel 3.1.4.1) [3]

- `AuthorizedKeysCommand`
  Hinterlegt einen Befehl, mit welchem die Public-Keys eines Benutzers geprüft werden können, wenn kein Treffer mittels `AuthorizedKeysFile` erzielt werden konnte
  Standardeinstellung: `none` (Kein definierter Befehl)
- `AuthorizedKeysCommandUser`
  Definiert den Benutzer, unter welchem der Befehl unter `AuthorizedKeysCommand` ausgeführt wird
  Standardeinstellung: `none` (Kein definierter Benutzer)
- `AuthorizedKeysFile`
  Zeigt auf die **Datei mit Public-Keys**, welche **zur Public-Key-Authentisierung** verwendet werden soll
  Standardeinstellung: `.ssh/authorized_keys` (Pfad relativ zum Benutzer-Heimverzeichnis, wobei mehrere und/oder absolute Pfade auch möglich sind)
- `PubkeyAcceptedAlgorithms`
  Spezifiziert die erlaubten **Algorithmen für die Public-Key-Authentisierung**
  Standardeinstellung:
  `ssh-ed25519-cert-v01@openssh.com,`
  `ecdsa-sha2-nistp256-cert-v01@openssh.com,`
  `ecdsa-sha2-nistp384-cert-v01@openssh.com,`
  `ecdsa-sha2-nistp521-cert-v01@openssh.com,`
  `sk-ssh-ed25519-cert-v01@openssh.com,`
  `sk-ecdsa-sha2-nistp256-cert-v01@openssh.com,`
  `rsa-sha2-512-cert-v01@openssh.com,`
  `rsa-sha2-256-cert-v01@openssh.com,`
  `ssh-ed25519,ecdsa-sha2-nistp256,`
  `ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,`
  `sk-ssh-ed25519@openssh.com,`
  `sk-ecdsa-sha2-nistp256@openssh.com,`
  `rsa-sha2-512,rsa-sha2-256`
- `PubkeyAuthOptions`
  Legt die zusätzlichen **Optionen zur Public-Key-Authentisierung** fest
  Standardeinstellung: `none`:
  - `none`: Keine zusätzlichen Optionen definiert
  - `touch-required`: Fordert bei Algorithmen für FIDO Authenticators (z.B. `ecdsa-sk` oder `ed25519-sk`) eine physische Benutzerbestätigung, was bei einem **FIDO** Authenticator in Hardware-Form durch Berührung geschieht
    Dies wird standardmässig gefordert, kann jedoch mit einer Option im `AuthorizedKeysFile` überschrieben werden
    Mit dieser Option wird die Eigenschaft entsprechend forciert
  - `verify-required`: Benötigt bei Algorithmen für **FIDO** Authenticators, dass der Benutzer z.B. mittels PIN verifiziert wird
- `PubkeyAuthentication`
  Aktivierung/Deaktivierung der **Public-Key-Authentisierung**
  Standardeinstellung: `yes` (Aktiviert)
- `RevokedKeys`
  Zeigt auf eine Datei, wessen Inhalt **revozierte Public-Keys** beinhaltet, welche für die Public-Key-Authentisierung nicht zugelassen sind
  Die Angabe im Format von mit `ssh-keygen` erstellten „Key Revocation Lists“ (siehe Kapitel 3.4.2.4) ist ebenfalls möglich
  Kann die Datei nicht gelesen werden, wird jeder Key zur Public-Key-Authentisierung abgelehnt
  Standardeinstellung: `none`
- `SecurityKeyProvider`
  Ermöglicht die Angabe einer spezifischen **Programmbibliothek**, welche zum Zugriff auf Keys von **FIDO Authenticators** verwendet werden soll
  Standardeinstellung: Keine Angabe

Authentisierung und Autorisierung: Zertifikats-Authentisierung [3]

- `AuthorizedPrincipalsCommand`
  Weist auf einen Befehl hin, mit welchem eine Liste erlaubter „Principals“[13] wie unter `AuthorizedPrincipalsFile` generiert wird
  Standardeinstellung: `none` (Kein definierter Befehl)
- `AuthorizedPrincipalsCommandUser` Definiert den Benutzer, unter welchem der Befehl unter `AuthorizedPrincipalsCommand` ausgeführt wird
  Standardeinstellung: `none` (Kein definierter Benutzer)
- `AuthorizedPrincipalsFile`
  Spezifiziert die Datei, welche bei der **Zertifikats-Authentisierung erlaubte „Principals“**[13] enthält
  Standardeinstellung: `none` (Keine definierte Datei)
- `TrustedUserCAKeys`
  Beschreibt eine **Datei mit** Public-Keys von **CAs**, dessen Zertifikate für die Authentisierung zugelassen sind
  Zugehörige Zertifikate ohne „Principals“[13] werden nicht zugelassen
  Standardeinstellung: Keine Definition

Benutzer und Gruppen [3]

- `AllowGroups`
  Ermöglicht die **Einschränkung des Logins** auf Benutzer, die zu einer **mit Namen passenden Gruppe** gehören
  Trifft eine Gruppe auch bei der Option `DenyGroups` zu, wird der Login trotzdem abgelehnt
  Muster / „Patterns“ gemäss Manpage der Client-Konfigurationsdatei [57] sind möglich
  Standardeinstellung: Alle Gruppen sind zugelassen
- `AllowUsers`
  Definiert **erlaubte Benutzer anhand ihres Benutzernamens** analog zur Option `AllowGroups`
  Trifft ein Benutzername auch bei der Option `DenyUsers` zu, wird der Login trotzdem abgelehnt
  Standardeinstellung: Alle Benutzer sind zugelassen
- `DenyGroups`
  Die Angabe **nicht erlaubter Gruppennamen** erfolgt analog zu `AllowGroups`, wobei dieser Parameter gegenüber `AllowGroups` Vorrang hat
  Standardeinstellung: Alle Gruppen sind zugelassen
- `DenyUsers`
  Die Auflistung **nicht zulässiger Benutzernamen** erfolgt analog zu `AllowUsers`
  Dieser Parameter hat Priorität gegenüber `AllowUsers`
  Standardeinstellung: Alle Benutzer sind zugelassen
- `PermitRootLogin`
  Spezifiziert, ob ein **Login mit dem Benutzer** `root` erfolgen darf, wobei es folgende 4 Optionen gibt:
  Standardeinstellung: `prohibit-password`
  - `yes`: `root`-Benutzer darf sich einloggen
  - `prohibit-password`: `root` darf sich einloggen, sofern er dazu keine „Keyboard-Interactive“- oder Passwort-Authentisierung verwendet
  - `forced-commands-only`: `root` darf sich nur mit der Public-Key-Authentisierungsmethode einloggen, aber nur mit definierter `Command`-Option [47]
  - `no`: `root` darf sich nicht einloggen

[13] Ein „Principal“ ist eine Bezeichnung, welche bei der Zertifikatserstellung zu einem Schlüssel (z.B. mit `ssh-keygen -n` [53]) mitgegeben werden kann, wobei die Angabe mehrerer „Principals“ möglich ist [68]

Forwarding / Weiterleitungen [3]

- Betreffend Local Forwarding und Remote Forwarding:
  - Mittels **Local Forwarding** kann ein Port oder Unix Socket von einem Client auf einen Server weitergeleitet werden
    Der Client hört auf einem bestimmten Port oder Socket und leitet die entsprechende Verbindung an den Server weiter, welcher diese zu einem entsprechenden Port oder Socket weiterleitet [69]
    Beispiel [69]: `ssh -L 80:www:80 server`

Abbildung 3.1.: Local Forwarding mit `ssh -L 80:www:80 server`

Mit `GatewayPorts` in der SSH-Client-Konfigurationsdatei kann definiert werden, ob der abzuhörende Port auf dem Client von anderen Hosts (`yes`) oder nur über die lokale Loopback-Adresse (`no`) erreichbar sein soll [57].

  - Mittels **Remote Forwarding** kann ein Port oder Unix Socket von einem Server auf einen Client weitergeleitet werden
    Der Server hört auf einem bestimmten Port oder Socket und leitet die entsprechende Verbindung an den Client weiter, welcher diese zu einem entsprechenden Port oder Socket weiterleitet [69]
    Beispiel [69]: `ssh -R 8080:localhost:80 server`

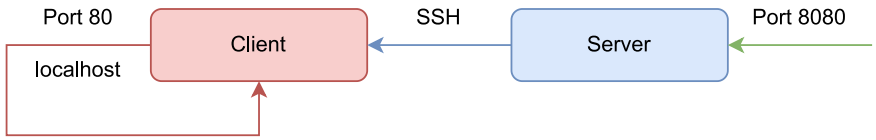

Abbildung 3.2.: Remote Forwarding mit `ssh -R 8080:localhost:80 server`

Mit `GatewayPorts` in der SSH-Server-Konfigurationsdatei kann definiert werden, ob der abzuhörende Port auf dem Server von anderen Hosts (`yes`) oder nur über die lokale Loopback-Adresse (`no`) erreichbar sein soll [3].

- `AllowAgentForwarding`
  Aktiviert/Deaktiviert die Verwendung des auf dem Client ausgeführten **Authentisierungs-Agenten** `ssh-agent`, sodass dieser für weitere SSH-Verbindungen ab dem verbundenen Server verwendet werden kann
  Standardeinstellung: `yes` (aktiviert)
- `AllowStreamLocalForwarding`
  Aktiviert/Deaktiviert das Local Forwarding und/oder Remote **Forwarding** von **Unix Sockets**
  Standardeinstellung: `yes` (aktiviert)
- `AllowTcpForwarding`
  Aktiviert/Deaktiviert das Local Forwarding und/oder Remote **Forwarding** von **TCP-Ports**
  Standardeinstellung: `yes` (aktiviert)
- `DisableForwarding`
  Unterbindet bei Aktivierung **sämtliche Weiterleitungsfunktionalitäten** und überschreibt dessen Optionen
  Standardeinstellung: `no`
- `GatewayPorts`
  Definiert, ob der abzuhörende Port auf dem Server oder Client bei Local oder Remote **Forwarding** von anderen Hosts unter der „**Wildcard**-Adresse“ (`yes`) **oder** nur über die lokale **Loopback**-Adresse (`no`) erreichbar sein soll
  Eine spezifische Angabe zur anzubindenden Adresse ist ebenfalls möglich
  Standardeinstellung: `no`
- `PermitListen`
  Legt fest, welche **Ports** auf dem Server für das **Remote Forwarding** geöffnet werden dürfen
  Mehrere Einträge möglich
  Standardeinstellung: `any` (Alle Ports erlaubt, wobei die Adress-Anbindung mit `GatewayPorts` eingeschränkt wird)
- `PermitOpen`
  Besagt welche **Ziel-Adressen und -Ports** vom Server aus für das **Local Forwarding** verwendet werden dürfen
  Standardeinstellung: `any` (Alle Ports erlaubt)
- `PermitTunnel`
  Spezifiziert, ob ein **Tunneling** (mit z.B. VPN) erlaubt ist, wobei die Optionen `point-to-point` (Layer 3, „Network Layer“[70]), `ethernet` (Layer 2, „Data Link Layer“[70]), `yes` (beide Varianten) oder `no` (kein Tunneling) zur Auswahl stehen
  Standardeinstellung: `no`
- `StreamLocalBindMask`
  Definiert die „file creation mode mask“ **umask**, mit welcher erstellte **Unix Socket-Dateien** fürs Local oder Remote Forwarding erstellt werden
  Standardeinstellung: `0177` (Unix Socket-Datei kann nur von Besitzer gelesen und geschrieben werden)
- `StreamLocalBindUnlink`
  Legt fest, ob bei einem **bereits existierenden Unix Socket** dieser fürs Local oder Remote Forwarding **entfernt und neu erstellt** werden soll
  Existiert bereits ein Unix Socket und diese Einstellung ist deaktiviert (`no`), kann die Weiterleitung nicht durchgeführt werden
  Standardeinstellung: `no`
- `X11DisplayOffset`
  Beschreibt die **erste Display-Nummer**, welche für die **X11-Weiterleitung** verwendet werden kann, um nicht die „realen“ X11-Server (für die Anzeige direkt auf dem Server-Host) zu beeinträchtigen
  Standardeinstellung: `10`
- `X11Forwarding`
  Aktiviert/Deaktiviert die **X11-Weiterleitung**
  Standardeinstellung: `no` (Deaktiviert)
- `X11UseLocalhost`
  Definiert die **Anbindung** des **X11-Forwarding-Servers** an die lokale **Loopback-Adresse**, um Zugriffe von entfernten Hosts auf den Proxy Display zu unterbinden
  Alternativ kann aus Kompatibilitätsgründen eine Anbindung an die Wildcard-Adresse erfolgen
  Standardeinstellung: `yes`
- `XAuthLocation`
  Zeigt auf den absoluten Pfad der `xauth`-Applikation, mit welcher Informationen zur Autorisierung betreffend X11-Server angezeigt und editiert werden können
  Standardeinstellung: `/usr/X11R6/bin/xauth`

Logging [3]

- `LogLevel`
  Definiert den **Detailgrad** („Log-Level“ bzw. „verbosity level“), mit welchem Nachrichten von `sshd` **geloggt** werden
  Eine Beschreibung der Loglevels kann der `syslog`-Manpage [71] entnommen werden, wobei die `sshd_config`-Manpage [3] die möglichen Levels auflistet
  Standardeinstellung: `INFO`
- `LogVerbose`
  Spezifiziert Muster zum `sshd`-Quellcode (Quelldatei, Funktion und/oder Zeilennummer), mit welchen bestimmte Ausgaben von Funktionen trotz anders definiertem `LogLevel` geloggt werden können (zur **Debugging**-Unterstützung)
  Standardeinstellung: Keine Definition
- `SyslogFacility`
  Definiert das **Merkmal** (**„Log-Facility“**), unter welchem Nachrichten von `sshd` **geloggt** werden
  Eine Beschreibung der Facilities kann der `syslog`-Manpage [71] entnommen werden, wobei die `sshd_config`-Manpage [3] die möglichen Facilities auflistet
  Standardeinstellung: `AUTH`

Maximalwerte und Timing-spezifisches [3]

- `ChannelTimeout`
  Spezifiziert die Dauer bis **inaktive SSH-Kanäle** geschlossen werden
  Die Dauer kann je nach Kanal-Typ (z.B. `session:shell` oder `forwarded-tcpip`) definiert werden
  Standardeinstellung: `none` (Kein Timeout)
- `ClientAliveCountMax`
  Legt die maximal mögliche Anzahl an **„Client-Alive“-Meldungen** fest, die ein Client gemäss definiertem `ClientAliveInterval` auslassen kann, **bevor dessen Verbindung getrennt** wird
  Standardeinstellung: `3` (nach 3 * `ClientAliveInterval` ohne „Client-Alive“-Meldung wird die Verbindung getrennt)
- `ClientAliveInterval`
  Definiert einen Intervall in Sekunden, bei dessen Ablauf ohne erhaltene Meldung **vom Client eine Antwort** von diesem **angefordert** wird
  Standardeinstellung: `0` (Funktionalität deaktiviert)
- `LoginGraceTime`
  Beschreibt die Dauer in Sekunden, nach welcher die **Verbindung ohne erfolgreiches Einloggen getrennt** werden soll
  Standardeinstellung: `120`
- `MaxAuthTries`
  Definiert die maximale **Anzahl an Authentisierungsversuchen** bevor die Verbindung getrennt wird
  Ab der Hälfte des angegebenen Wertes werden zusätzliche Fehler geloggt
  Standardeinstellung: `6`

- `MaxSessions`
  Spezifiziert der Maximalwert bezüglich **offener Sessions pro Verbindung**, wobei ein Wert von `1` das Session-Multiplexing (siehe Kapitel 3.1.5) deaktiviert und ein Wert von `0` alle Sessions verhindert während gleichzeitig Weiterleitungen erlaubt sind
  Standardeinstellung: `10`
- `MaxStartups`
  Legt die maximal mögliche Anzahl **gleichzeitiger, nicht-authentisierter Verbindungen** zum SSH-Daemon fest, bei welcher zusätzliche Verbindungsversuche abgeworfen werden
  Angabe mit 3 durch Doppelpunkt getrennte Zahlen (`start:rate:full` [14])
  Nicht-authentisierte Verbindungen können mittels `LoginGraceTime`-Option früher geschlossen werden
  Standardeinstellung: `10:30:100`
- `PerSourceMaxStartups`
  Besagt das Limit der maximalen **nicht-authentisierten Verbindungen pro Quelladresse**, wobei der niedrigere Wert zwischen diesem Parameter und `MaxStartups` angewendet wird
  Standardeinstellung: `none` (Kein Limit)
- `PerSourceNetBlockSize`
  Beschreibt die Anzahl der angewandten **Bits**, ab welchen **Quelladressen** bei `PerSourceMaxStartups` **gruppiert** werden
  Ermöglicht es den Parameter `PerSourceMaxStartups` auf Subnetze anstelle einzelner IP-Adressen anzuwenden
  Standardeinstellung: `32:128` (komplette IPv4-Adresse (32 Bit [72]) und komplette IPv6-Adresse (128 Bit [73]))
- `RekeyLimit`
  Definiert die **Maximalmenge an Daten**, die gesendet oder empfangen werden dürfen, **bevor ein neuer Session-Key ausgehandelt wird** (siehe Liste in Kapitel 3.4.3.1, Punkt 1c)
  Die Angabe kann in `K` für Kilo-, `M` für Mega- oder `G` für Gigabytes erfolgen
  Standardeinstellung: `default none` (Der Standardwert entspricht zwischen `1G` oder `4G`, je nach verwendetem Verschlüsselungsalgorithmus)
- `TCPKeepAlive`
  Aktiviert/Deaktiviert das Senden von **„TCP-keepalive“-Nachrichten**, mit welchen SSH-Verbindungen bei einem Netzwerkunterbruch detektiert und beendet werden können (Alternativ würden Sessions hängig bleiben)
  Standardeinstellung: `yes` (Aktiviert)
- `UnusedConnectionTimeout`
  Beschreibt wie schnell Verbindungen ohne offene Kanäle geschlossen werden sollen
  Durch Remote Forwarding geöffnete Ports (`ssh -R`) zählen hierbei nicht als offene Kanäle und verhindern somit nicht den Timeout
  Standardeinstellung: `none` (Kein Timeout)

[14] Ab `start` Anzahl nicht-authentisierter Verbindungen werden neue Verbindungen mit einer Wahrscheinlichkeit von `rate` (in Prozent) getrennt, wobei die Wahrscheinlichkeit linear steigt bis die Anzahl nicht-authentisierter Verbindungen bei `full` liegt und alle Versuche verweigert werden

Netzwerkkonfiguration [3]

- `AddressFamily`
  Wahl der zu verwendenden **IP-Adressfamilie** (IPv4 und/oder IPv6)
  Standardeinstellung: `any` für IPv4 und IPv6
- `IPQoS`
  Ermöglicht die Definition der zu verwendenden Werte für den IPv4-Parameter **„type-of-service“** bzw. DCSP (Differentiated Services Code Points)
  Standardeinstellung: `af21 cs1`
  „Low-Latency Data“ [74] für interaktive Sessions (`af21`)
  „Low-Priority Data“ [74] bzw. „Lower Effort“ für nicht-interaktive Sessions (`cs1`)
- `ListenAddress`
  Spezifiziert die **Adresse** und ggf. den **Port**, auf welchem `sshd` für Verbindungen empfänglich sein soll
  Mehrere Einträge sind möglich
  Ohne Angabe eines Ports hört der Server auf allen mit `Port` definierten Ports zu den spezifizierten Adressen
  Die Angabe von rdomains (Routing Domains) ist ebenfalls möglich
  Standardeinstellung:
  `ListenAddress [::]:22`
  `ListenAddress 0.0.0.0:22`
- `Port`
  Definiert die **Port-Nummer**, auf welcher der SSH-Daemon auf Verbindungen wartet
  Die Option `ListenAddress` ist hier ebenfalls zu beachten
  Mehrere Einträge sind möglich
  Standardeinstellung: `22` [75]
- `RDomain`
  Legt eine **explizite rdomain** fest, zu welcher die Session sowie weitergeleitete Ports oder Unix Sockets angebunden werden
  Mit der Angabe von `\%D` wird die rdomain, in welcher die SSH-Verbindung eingegangen ist, angebunden
  Standardeinstellung: Keine Angabe

Server-Schlüssel und -Zertifikate [3]

- `HostCertificate`
  Zeigt auf ein **Zertifikat**, wessen Public-Key dem durch `HostKey` festgelegten Private-Key entsprechen muss
  Standardeinstellung: Kein Zertifikat

- `HostKey`
  Definiert die vom **Server** zu verwendenden **Private-Keys**, wobei die möglichen Schlüssel anhand des definierten Algorithmus gemäss `HostKeyAlgorithms` gewählt werden müssen
  Mehrere Schlüssel sind durch mehrmalige Angabe definierbar (pro Algorithmus ein Schlüssel)
  Bei Verwendung eines Authentisierungs-Agenten, der den Private-Key besitzt, ist auch die Angabe des zugehörigen Public-Keys möglich
  In OpenBSD werden die HostKeys automatisch unter den Standardpfaden generiert mit `ssh-keygen -A` [53]
  Standardeinstellung:
  `HostKey /etc/ssh/ssh_host_ecdsa_key`
  `HostKey /etc/ssh/ssh_host_ed25519_key`
  `HostKey /etc/ssh/ssh_host_rsa_key`

- `HostKeyAgent`
  Spezifiziert den **Unix Socket**, über welchen mit dem **Authentisierungs-Agenten** mit Zugriff zu den Private-Keys kommuniziert wird
  Ist die Umgebungsvariable `SSH_AUTH_SOCK` definiert, wird der Socket daraus gelesen[15]
  Standardeinstellung: `none`

- `HostKeyAlgorithms`
  Definiert die vom Server offerierten **Algorithmen zum Signieren des Host-Schlüssels**
  Standardeinstellung: `ssh-ed25519-cert-v01@openssh.com,ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com,sk-ssh-ed25519-cert-v01@openssh.com,sk-ecdsa-sha2-nistp256-cert-v01@openssh.com,rsa-sha2-512-cert-v01@openssh.com,rsa-sha2-256-cert-v01@openssh.com,ssh-ed25519,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,sk-ssh-ed25519@openssh.com,sk-ecdsa-sha2-nistp256@openssh.com,rsa-sha2-512,rsa-sha2-256`

Umgebungsvariablen [3]

- `AcceptEnv`
  Definition **erlaubter Umgebungsvariablen**, die vom Client definiert werden
  Standardeinstellung: Keine, ausser `TERM` bei Pseudo-Terminals

- `PermitUserEnvironment`
  Besagt, ob **Benutzerspezifische Umgebungsvariablen** geladen werden (siehe Liste in Kapitel 3.4.3.1, Punkt 5d)
  Angabe in Mustern zu erlaubten Variablennamen ist möglich
  Standardeinstellung: `no`

- `PermitUserRC`
  Spezifiziert, ob das **RC-Skript** `~/.ssh/rc` ausgeführt wird (siehe Liste in Kapitel 3.4.3.1, Punkt 5f)
  Standardeinstellung: `yes`

- `SetEnv`
  Ermöglicht das Festlegen bestimmter Umgebungsvariablen und **überschreibt** die Standard-**Umgebungsvariablen** sowie auch welche, die mit `AcceptEnv` und `PermitUserEnvironment` definiert wurden
  Standardeinstellung: Keine Definition

[15] Die Variable `SSH_AUTH_SOCK` wird beim Start des Authentisierungs-Agenten `ssh-agent` gefüllt [56]

Weitere Einstellungen [3]

- `Banner`
  Der **Inhalt** dieser mittels absolutem Pfad hinterlegten Datei wird **vor der Authentisierung angezeigt** (siehe Kapitel 3.1.4)
  Standardeinstellung: `none` (Keine Datei)
- `ChrootDirectory`
  Definiert **ob und welches Verzeichnis als** neues Wurzel- bzw. **„root"-Verzeichnis** verwendet werden soll
  Das „root"-Verzeichnis muss über entsprechende Dateien und Ordner für die Benutzersession verfügen, was u. a. eine Shell sowie spezifische `/dev`-Nodes wie `stdin` und `stdout` beinhaltet (bei Sitzungen mit SFTP sind diese Anforderungen nicht nötig)
  Standardeinstellung: `none` („root"-Verzeichnis nicht wechseln)
- `Compression`
  Legt fest, ob eine **Kompression** nach der erfolgreichen Authentisierung möglich ist
  Standardeinstellung: `yes`
- `ExposeAuthInfo`
  Schreibt bei Aktivierung **in eine temporäre Datei Informationen zur erfolgten Authentisierung** der entsprechenden Session
  Der Dateipfad wird mittels Umgebungsvariable `SSH_USER_AUTH` definiert
  Standardeinstellung: `no`
- `ForceCommand`
  Ermöglicht die **Forcierung zur Ausführung eines Befehls** und ignoriert die vom Client angefragte Befehle oder Benutzerspezifische „RC-Skripte" (siehe Liste in Kapitel 3.4.3.1, Punkte 5f und 5g)
  Standardeinstellung: `none` (Kein Befehl)
- `Include`
  Zeigt auf **Dateien, welche in die SSH-Server-Konfiguration miteinbezogen** werden sollen
  Relative Pfade werden von `/etc/ssh` aus betrachtet, wobei die Angabe mehrerer Dateien und Wildcards („*") möglich sind
  Standardeinstellung: Keine Datei angegeben
- `Match`
  Führt ein **Block mit Bedingungen („conditional block"** / „`Match`-Block") ein, wessen Parameter bei Erfüllung der Bedingung die entsprechenden, allgemein definierten Parameter ersetzen
  Dazu werden sämtliche Optionen unterhalb des zutreffenden `Match`-Parameters übernommen bis der nächste `Match`-Eintrag oder das Ende der Konfigurationsdatei angetroffen wird
  Mögliche Bedingungen sind **u. a. Benutzername, Gruppe, Hostname oder Adresse**, wobei diese **mit Mustern** / „Patterns " [57] **beschrieben** werden können
  Treffen mehrere `Match`-Einträge zu, wird nur der erste `Match`-Block angewendet
  Innerhalb definierbare Optionen sind der entsprechenden Manpage [3] zu entnehmen
  Standardeinstellung: Keine Definition
- `PermitTTY`
  Beschreibt, ob sich der Benutzer einen **Pseudo-Terminal** allozieren darf
  Standardeinstellung: `yes`
- `PidFile`
  Legt den **Dateipfad** fest, unter welchem die **Prozess-Identifikationsnummer** von `sshd` geschrieben wird
  Standardeinstellung: `/var/run/sshd.pid`
- `PrintLastLog`
  Definiert, ob der **Zeitstempel des letzten Benutzerlogins** bei einer interaktiver Sitzung angezeigt werden soll (siehe Liste in Kapitel 3.4.3.1, Punkt 5a)
  Standardeinstellung: `yes`
- `PrintMotd`
  Spezifiziert, ob der Inhalt der Datei `/etc/motd` **(„Message of the day"**) beim Login zu einer interaktiver Sitzung darzustellen ist (siehe Liste in Kapitel 3.4.3.1, Punkt 5a)
  Standardeinstellung: `yes`

- `StrictModes`
  Besagt, ob die **Berechtigungen der Benutzer-Dateien und dessen Heimverzeichnis** vor dem Login **geprüft** werden sollen
  Mit `ChrootDirectory` definierte Verzeichnisse werden unabhängig von dieser Einstellung geprüft
  Standardeinstellung: `yes`
- `Subsystem`
  Konfiguriert externe **Subsysteme** (Name und Befehl), welche bei Anfrage ausgeführt werden können (siehe Kapitel 3.1.5.1)
  Die Angabe eines „In-Prozess"-SFTP-Servers ist mittels `internal-sftp` möglich, mit welchem Konfigurationen mit der Option `ChrootDirectory` einfacher ausfallen
  Standardeinstellung [16]:
  `sftp /usr/libexec/sftp-server`
- `UseDNS`
  Beschreibt, ob der Hostname des Clients geprüft werden soll und ob die ermittelte IP-Adresse der Client-IP-Adresse entspricht
  Ermöglicht das verwenden von Hostnamen in der Konfiguration (u. a. `Match`-Parameter)
  Standardeinstellung: `no` (Nur Adressen und keine Hostnames verwenden)
- `VersionAddendum`
  Spezifiziert einen Text, welcher beim „SSH-Protokoll-Banner" (siehe „Identifikations-String" in Kapitel 3.1.3) angefügt werden soll
  Standardeinstellung: `none` (Kein Zusatz)

3.4.3.2. SFTP-Server-Subsystem sftp-server

Die Serverseite, welche das Protokoll SFTP spricht, erscheint in Form der Anwendung `sftp-server`, welche über `sshd` mit der `Subsystem`-Option aufzurufen ist [55]. Die Anwendung verfügt über Kommandozeilen-Parameter, welche ebenfalls via `Subsystem`-Option mitzugeben sind [55].

Unter anderem kann eine Komma-separierte Liste an SFTP-Requests (siehe „Interactive Commands" in der SFTP-Client-Manpage) mitgegeben werden, welche explizit erlaubt (`-p`) oder verboten (`-P`) sind [55]. Um zum Beispiel das Erstellen und Entfernen von Verzeichnissen zu unterbinden wäre der Parameter `-P mkdir,rmdir` anzugeben. Ein „Read-Only"-Mode (`-R`), welcher Schreibzugriffe unterbindet, kann aktiviert werden [55]. Die Definition eines Startverzeichnisses alternativ zum Benutzer-Heimverzeichnis kann mit `-d` angegeben werden [55]. Fürs Logging lassen sich „Log-Facility" (`-f`) und „Log-Level" (`-l`) mit den entsprechenden Parameter definieren [55].

```
1  ...
2  debug2: load_server_config: filename /etc/ssh/sshd_config
3  ...
4  debug3: /etc/ssh/sshd_config:86 setting Subsystem sftp  /usr/libexec/sftp-
       server
5  ...
6  debug1: server_input_channel_req: channel 0 request subsystem reply 1
7  debug1: session_by_channel: session 0 channel 0
8  debug1: session_input_channel_req: session 0 req subsystem
9  debug2: subsystem request for sftp by user user
10 debug1: subsystem: exec() /usr/libexec/sftp-server
11 debug2: channel_set_xtype: labeled channel 0 as session:subsystem:sftp (
       inactive timeout 0)
12 Starting session: subsystem 'sftp' for user from 192.168.100.1 port 7718 id 0
13 ...
14 Received disconnect from 192.168.100.1 port 7718:11: disconnected by user
15 Disconnected from user user 192.168.100.1 port 7718
```

Quelltext 3.7: SSH-Server-Debugging-Meldungen während Ausführung von sftp auf dem Client

[16] Gemäss `sshd_config`-Manpage ist kein Subsystem als Standard definiert, jedoch wird diese Einstellung in der Standardausführung der Konfiguration überschrieben

3.4.3.3. Authentisierungs-Agent ssh-agent

Der Authentisierungs-Agent `ssh-agent` kann verwendet werden, um die Private-Keys für die Public-Key-Authentisierung zu laden [56]. Durch zugehörige Umgebungsvariablen kann der Agent lokalisiert und automatisch für die Authentisierung via SSH verwendet werden [56].

Private-Key-Identitäten können `ssh-agent` mit der Anwendung `ssh-add` (siehe Kapitel 3.4.2.1) oder mittels SSH-Client mit aktivierter Option `AddKeysToAgent` hinzugefügt werden [56].

Eine Vielzahl von Parametern können `ssh-agent` mitgegeben werden, wobei u. a. folgende Optionen zur Auswahl stehen (mit Angabe über die Kommandozeile) [56]:

Beschreibung	Kommandozeile
Anwendung **nicht im Hintergrund** ausführen Deaktivierung des Daemon-Verhaltens	`-D`
Wahl des **Hash-Algorithmus** zur Darstellung des **Key-Fingerprints** Standardeinstellung: `sha-256`	`-E sha256`
Beenden des derzeit ausgeführten Agenten gemäss Prozess-Identifikationsnummer Umgebungsvariable `SSH_AGENT_PID`	`-k`
Definition von Muster betreffend erlaubter Pfade zu **PKCS#11-Provider oder FIDO-Authenticator-Programmbibliotheken** Standardeinstellung: `/usr/lib/*,/usr/local/lib/*`	`-P`
Spezifizierung der maximalen Lebensdauer von hinzugefügten Identitäten Wurde eine Lebensdauer beim Hinzufügen mittels `ssh-add` definiert, hat diese Vorrang	`-t`

Tabelle 3.4.: Auszug Authentisierungs-Agent-Konfigurationsparameter (Kommandozeile) [54]

Zusätzlich kann ein Befehl am Ende mitgegeben werden, welcher als Subprozess von `ssh-agent` ausgeführt wird und bei dessen Ende `ssh-agent` ebenfalls beendet wird [56].

Die Manpage von `ssh-agent` [56] empfiehlt dessen Start in einer Shell mittels folgenden Befehls: `eval $(ssh-agent -s)` [17].

[17] Alternative Schreibweise: `eval `ssh-agent -s`` [76]

3.5. Empfehlungen und „Best Practices“

Dieses Kapitel betrachtet diverse Empfehlungen, um sich einen Überblick über allgemein genutzte Konfigurationsoptionen sowie empfohlene Algorithmen zu verschaffen. Dazu werden Vorgaben bzw. Standards diverser Institute, in Kapitel 3.1 referenzierte RFC-Dokumente sowie andere Quellen zu Rate gezogen. Hierbei handelt es sich um keine abschliessende Auflistung, jedoch soll durch das Hervorheben der Gemeinsamkeiten generell anerkannte Empfehlungen ermittelt werden.

Es wird jeweils davon ausgegangen, dass passende Bedingungen [18] geben sind, die jeweiligen Komponenten angemessen eingesetzt und geheimzuhaltende Daten wie Private-Keys entsprechend geschützt werden.

Sollten für den Einsatz von SSH je nach Verwendungszweck bestimmte Rahmenbedingungen und ggf. Algorithmen oder Protokolle vorgegeben sein, sind diesen zur dessen Erfüllung Vorrang zu gewähren.

Obwohl die Publikationen Empfehlungen für einen längerfristigen Einsatz geben, können unvorhergesehene Änderungen eintreffen, welche z.B. die Empfehlung zur Verwendung eines bestimmten Algorithmus massgebend ändern kann. Daher wird zusätzlich vor dem SSH-Einsatz empfohlen, den aktuellsten Stand der Technik zu prüfen [19].

Aus der Kombination diverser Publikationen und Artikel ergibt sich ein Minimalstandard betreffend anzuwendender Algorithmen und Protokolle sowie dem Umgang mit SSH, welcher in Kapitel 3.5.3 festgehalten wird.

3.5.1. Publikationen von Bundesbehörden

Publikationen von Bundesbehörden geben Punkte für entsprechende IT-Infrastrukturen vor, welche dessen Aufbau und Betrieb beeinflussen. Sie werden regelmässig überarbeitet und beruhen auf anerkannten Quellen (teilweise auf Publikationen anderer Bundesbehörden), welche die entsprechenden Funktionsweisen beweisen und ihre Empfehlungen begründen.

3.5.1.1. IT-Grundschutz der Schweizer Bundesverwaltung

Das Dokument „SiO01 Version 5.0“ *IT-Grundschutz in der Bundesverwaltung* [77] des Nationalen Zentrum für Cybersicherheit NCSC legt die Sicherheitsvorgaben für Informatikmittel in der Schweizer Bundesverwaltung fest und fordert, dass administrative Zugriffe und Fernzugriffe über das Netzwerk z.B. mit SSH kryptografisch abzusichern sind und dem aktuellen Stand der Technik entsprechen [77]. Zu diesem Technikstand wird auf ein Dokument aus dem Jahr 2023 mit Empfehlungen zu kryptografischen Verfahren [78] verwiesen, welche in naher Zukunft als sicher betrachtet werden können und für den Einsatz von maximal 10 Jahren gelten [78].

Im Dokument zu den kryptografischen Verfahren [78] werden konkrete Werte für SSH-Optionen (siehe Abschnitt „Algorithmen-Wahl“ in Kapitel 3.4.3.1) empfohlen, wobei die Wahl der Schlüsselaustausch-Algorithmen mit spezifischen Diffie-Hellman-Gruppen (`KexAlgorithms`, siehe Kapitel 3.1.3.3) hervorzuheben ist [78]:
`diffie-hellman-group15-sha512, diffie-hellman-group16-sha512,curve25519-sha256, ecdh-sha2-secp256r1,ecdh-sha2-secp348r1,ecdh-sha2-secp521r1` [20]

Weiterhin wird ein Dokument des BSI (Bundesamt für Sicherheit in der Informationstechnik) [79] referenziert, welches im folgenden Kapitel behandelt wird.

[18] Qualität Zufallszahlengenerator, Kompromittierung der Systeme, Fehlerfreie Implementierungen, etc.

[19] In dieser Arbeit wird der aktuellste Stand der Technik zwar geprüft, jedoch kann sich dieser nach Arbeits-Abschluss bereits geändert haben

[20] Wie bereits in Kapitel 3.1.3.3 erwähnt, handelt es sich bei `secp256r1`, `secp384r1` und `secp521r1` um die gleichen Kurven wie `NIST P-256`, `NIST P-384` und `NIST P-521`, es wurde der gleichen Kurve von verschiedenen Organisationen einen anderen Namen gegeben [30]

3.5.1.2. BSI (Deutsches Bundesamt)

Das BSI (Bundesamt für Sicherheit in der Informationstechnik) hat eine technische Richtlinie „BSI TR-02102-1“ *Kryptographische Verfahren: Empfehlungen und Schlüssellängen* [79] veröffentlicht, welches die Sicherheit ausgewählter kryptografischer Verfahren für die Einführung neuer Systeme ab 2023 bewertet [79].

Hierbei handelt es sich um ein ausführliches Dokument, welches die vorgeschlagenen Mechanismen detailliert beschreibt und begründet sowie auch auf das Generieren von Zufallszahlen eingeht. Im Vergleich zum referenzierten Empfehlungsdokument des IT-Grundschutz [78] werden weitere Algorithmen (u. a. die Verschlüsselungsalgorithmen ECIES und DLIES) aufgeführt, wobei es auch welche gibt, die im BSI-Dokument [79] nicht vorzufinden sind. Anstelle von Minimalgrössen wie 3072 oder 255 Bits werden analog die Grössen 3000 und 250 Bits vorgegeben [79].

3.5.1.3. NIST (US-amerikanische Behörde)

Eine Publikation mit genehmigten Algorithmen der NIST (National Institute of Standards and Technology) für dessen Bundesregierung wurde unter „NIST SP 800-175B Rev. 1“ *Guideline for Using Cryptographic Standards in the Federal Government: Cryptographic Mechanisms* veröffentlicht [80]. Es tritt in ähnlicher Ausführlichkeit zum BSI-Dokument [79] auf und verweist auf weitere NIST-Dokumente wie z.B. „NIST SP 800-131A Rev. 2“ [20] oder „FIPS 186-5“ [17] [80].

3.5.2. Weitere Artikel und Publikationen

Artikel aus dem Internet gehen gegenüber Dokumenten der Bundesbehörden gezielter auf den Gebrauch von SSH ein, wobei die Publikationen der Bundesbehörden vor dessen Veröffentlichung genauer und von mehreren Parteien geprüft werden und veraltete Versionen entsprechend kennzeichnen, was am Beispiel von „NIST SP 800-131A Rev. 2“ [20] zu sehen ist [21]. Gleiches wie bei der referenzierten NIST-Publikation ist auch bei RFC-Dokumenten vorzufinden, welche auf Kommentare zum entsprechenden RFC in Form von Mail-Diskussionen verweist.

Einen Artikel im Internet kann jeder publizieren, weshalb die Webseite des Artikels, zugehörige Kommentare und entsprechende Autoren einen Einfluss auf dessen Glaubwürdigkeit haben. Aus diesem Grund werden Artikel von bekannten Webseiten gegenüber z.B. Forum-Einträgen stärker gewichtet. Unter anderem existieren Artikel auf Firmen-Webseiten, dessen Mitarbeiter an SSH-RFCs mitgewirkt haben (z.B. ssh.com [81]) oder dessen Produkte SSH verwenden (z.B. redhat.com [82]).

Die Organisation Center for Internet Security (CIS) veröffentlicht Empfehlungen zur Härtung diverser Produkte, darunter auch Linux-Distributionen, in Form von Dokumenten der Reihe „CIS Benchmarks“ [83]. Diese Dokumente für Linux-Distributionen (u.a. Debian 11 [84] und Red Hat Enterprise Linux 9 [85]) sowie Distributions-unabhängige Linux-Betriebssystem-Empfehlungen [86] beinhalten konkrete und begründete Empfehlungen zur SSH-Server-Konfiguration.

Bücher zu SSH wurden ebenfalls veröffentlicht, wobei die Titel *SSH, The Secure Shell: The Definitive Guide, 2nd Edition* (2005) [87] und *SSH Mastery - Second Edition* (2018) [60] mit guten Rezensionen herausstechen. Allgemeine IT-Bücher wie *Hacking & Security, 3., aktualisierte und erweiterte Auflage* (2022) [88] gehen ebenfalls auf das Protokoll ein. Aus Zeitgründen wurden lediglich besagte 3 Bücher zu rate gezogen, wobei es ziemlich sicher noch Weitere mit gleichwertigen oder besseren Bewertungen gibt. Betreffend Bücher gilt es zu bemerken, dass diese ähnlich wie z.B. RFC-Dokumente geprüft, jedoch weniger oft aktualisiert werden, weshalb der entsprechende Stand der Technik meist älter ist und ggf. erwähnte Algorithmen oder Protokolle allgemein nicht mehr verwendet werden sollen.

Für den ermittelten Minimalstandard in Kapitel 3.5.3 werden entspr. verwendete Quellen referenziert.

[21] Die Webseite zur verwiesenen NIST-Publikation zeigt welches Dokument durch dieses ersetzt wird, eine Versionierung mit Verweis zu alten Dokumenten und zur Publikation erhaltene Kommentare

3.5.3. Ermittelter Minimalstandard

Folgende Algorithmen und Protokolle werden in Kombination der betrachteten Publikationen als Minimalstandard empfohlen:

- **Symmetrische Verschlüsselung**
  - **AES** (AES-CBC oder AES-CTR) mit 128 Bit Schlüssellänge [22]
    und geeignetem **HMAC** („Encrypt-then-MAC“) [78][79][20][80]
  - AEAD Betriebsmodi (**AES-GCM** oder **AES-CCM**) [78][79][20][80]
- **Asymmetrische Verschlüsselung**
  - **RSA** mit OAEP-Padding und **3072 Bit** Schlüssellänge [78][79][20][80]
- **Hashfunktionen**
  SHA-2 und SHA-3 mit mindestens **256 Bit** Ausgabelänge [78][79][20][80]
- **Datenauthentifizierung**
  HMAC mit SHA-2 und SHA-3 [78][79][20][80][81]
- **Digitale Signaturen**
  - **RSA** mit **3072 Bit** Schlüssellänge [23] [78][79][20][80]
  - **ECDSA** und **EdDSA** [24] über elliptischer Kurve aus Menge von „Safe Curves“ [25] [89], Brainpool-Kurven [90] oder NIST-Kurven mit Bitlänge von min. 255 [91][78][79][20][80]
- **Schlüsselaustausch**
  - **DHKE** (authentisiert) mit **3072 Bit** Gruppengrösse [78][79][20][80]
    und folgenden Diffie-Hellman-Gruppen:
    `diffie-hellman-group15-sha512,diffie-hellman-group16-sha512` [21][78][27]
  - **ECDH** (authentisiert) über elliptischer Kurve aus Menge von „Safe Curves“ [25] [89], Brainpool-Kurven [90] oder NIST-Kurven mit Bitlänge von min. 255 [91][78][79][20][80]

[22] AES mit 256 Bit für Schutz gegen Attacken durch Quantencomputer empfohlen [78]
[23] DSA wird von „FIPS 186-5“ [17] nicht mehr zur Verwendung freigegeben, obwohl der Algorithmus in den Dokumenten der anderen Behörden noch aufgeführt wird
[24] Zu EdDSA gehören auch u. a. dessen Varianten Ed25519 und Ed448
[25] „Safe Curves“ werden vom schweizerischen Dokument „SiO01 Version 5.0“ [77] bevorzugt und beinhaltet elliptische Kurven, welche auch in den NIST-Dokumenten vorkommen

Zusätzlich zur angemessenen Handhabung der Private-Keys, entsprechender Härtung des Servers, Vergabe der Berechtigungen sowie Verwendung von Zusatzsoftware [26], ergeben sich zur Konfiguration des SSH-Servers folgende Empfehlungen:

- **Authentisierung**
  - 2-Faktoren-Authentisierung (2FA) verwenden [77][79][93][88]
  - Filtern zugelassener Benutzer und Gruppen [82][87][60]
  - Host-basierte Authentisierung deaktivieren [87][85]
  - Login für den Benutzer `root` deaktivieren [27] [82][88][85]
  - Nicht verwendete Authentisierungsmethoden deaktivieren [94]
  - Reine Password-Authentisierung deaktivieren (durch 2FA impliziert) [82][94][87][60]
  - Private-Keys ohne Passwörter nicht verwenden [28] und Passwörter nicht bei Private-Key hinterlegen, für automatisierte Abläufe wenn möglich einen Authentisierungs-Agenten verwenden [29] [87][60]
  - Public-Key-Authentisierung verwenden [82][87][88]
  - Zertifikate bei der Authentisierung verwenden [81][60]
- **Forwarding / Weiterleitungen**
  jeglicher Art deaktivieren, sofern sie nicht benötigt werden [30] [81][94][85]
- **Spezifische Konfiguration**
  - Konfigurationen, welche nur unter bestimmten Bedingungen anzuwenden sind, mittels `Match` (siehe Abschnitt „Weitere Einstellungen“ in Kapitel 3.4.3.1) einschränken [31] [94][60]
  - Optionen für bestimmte Public-Keys (sofern nötig) limitieren [94][87]

Wird der Port des SSH-Servers geändert, kann dieser von einem Angreifer mittels Port-Scanning trotzdem erkannt werden (reduziert jedoch die Log-Ausgaben, da Angreifer auch direkt zum Standart-Port verbinden) [60].

Gemäss diversen Internet-Publikationen kann beim Ausführen des SSH-Servers mit dem Diagnostik-Tool `strace` bei einer Passwort-Authentisierung das Passwort im Klartext ausgelesen werden [95], was unter einer aktuellen Debian-Installation reproduziert werden konnte. Unter OpenBSD gibt es anstelle von `strace` die Applikation `ktrace`, mit welcher ein Reproduzieren nicht möglich war oder weiteres Auseinandersetzen benötigt. Dies ist ein Grund zur Deaktivierung der Passwort-Authentisierung, sofern dies möglich ist. Zudem erinnert dies auch daran, dass einem bewusst sein soll, an welchen Orten man sein Passwort eingibt.

[26] Zum Beispiel mit der Software „Fail2Ban“ [92] können fehlgeschlagene Login-Versuche anhand von Logging-Dateien erkannt und entsprechende Zugriffe anhand der IP-Adresse mittels Host-Firewall automatisch blockiert werden [92][88]

[27] Falls die Verwendung des `root`-Kontos für automatisierte Tätigkeiten unumgänglich ist, gilt es dessen Zugang mittels `PermitRootLogin forced-commands-only` auf die Ausführung eines bestimmten Befehls sowie zur Authentisierung mittels Public-Key einzuschränken (siehe Abschnitt „Benutzer und Gruppen“ in Kapitel 3.4.3.1) [81]

[28] Hier sind Private-Keys zur Authentisierung gemeint, SSH-Server-Host-Keys werden üblicherweise ohne Passwort hinterlegt [60]

[29] Der Private-Key wird zu Beginn manuell zum Authentisierungs-Agenten hinzugefügt, kann danach jedoch automatisiert verwendet werden während dieser im Arbeitsspeicher des Agenten geladen ist (der Schlüssel ist dort unverschlüsselt und kann mit entsprechendem Zugriff auf dem entsprechenden Host ausgelesen werden, wobei Einstellungen zu Agent Forwarding beachtet werden müssen) [87][60]

[30] Bei nötigem Forwarding sind erlaubte Verbindungen zu hinterlegen und nicht erlaubte Verbindungen zu blockieren [60]

[31] Beispielsweise können so bestimmte Benutzer nur SFTP-Operationen in einem definierten Verzeichnis ohne Kommandozeilenzugriff eingerichtet werden [60]

3.5.4. Vergleich mit SSH-Server-Standardeinstellungen

Beim Vergleich des ermittelten Minimalstandards aus Kapitel 3.5.3 mit den Standardeinstellungen des OpenSSH-Serverdienstes `sshd` aus Kapitel 3.4.3.1 werden Unterschiede festgestellt, welche in den folgenden Abschnitten behandelt werden.

3.5.4.1. Algorithmen

Die Wahl der Algorithmen wird von den OpenSSH Entwicklern mit entsprechender Begründung in dessen Release Notes [63] getroffen, orientiert sich am aktuellen Stand der Technik und ändert sich mit der Zeit [63]. Sollte eine Infrastruktur bestimmten Richtlinien unterliegen, welche eine klare Auswahl an Algorithmen vorgibt, so gilt es diese manuell in der Konfiguration zu pflegen. Die von OpenSSH vorgegebene Algorithmen-Wahl kann in der Konfiguration mit der entsprechenden Option (z.B. `Ciphers` oder `PubkeyAcceptedAlgorithms`) jeweils komplett überschrieben, ergänzt oder einzelne Algorithmen entfernt werden [32] [3].

Die vordefinierte Minimal-Schlüssel-Grösse in Bits für RSA ist mit 1024 unter der Option `RequiredRSASize` im Vergleich mit dem Minimalstandard zu klein [3]. Bezüglich DHKE kann die Gruppengrösse durch die entsprechende Diffie-Hellman-Gruppe festgelegt werden [33] [26].

Zur Erfüllung des ermittelten Minimalstandards gilt es die Algorithmen-, Diffie-Hellman-Gruppen- und RSA-Schlüsselgrössen-Wahl entsprechend einzuschränken.

3.5.4.2. Authentisierung

Die Standardeinstellungen von `sshd` erlauben eine einfache Passwortauthentisierung und filtern weder nach Benutzer noch Gruppen [3]. Lediglich der Benutzer `root` wird standardmässig eingeschränkt, indem für dieser kein Login mittels Passwort erfolgen darf [3].

Als Authentisierungsmethoden sind sämtliche erlaubt, wobei der SSH-Server mit aktivierter „Keyboard-Interactive“-, Passwort- und Public-Key-Authentisierung ausgeliefert wird [3].

Nach ermitteltem Minimalstandard ist es demnach noch nötig eine 2-Faktoren-Authentisierung einzurichten, den Zugriff nur für erlaubte Benutzer und/oder Gruppen einzuschränken, den `root`-Login zu deaktivieren und Public-Key-Authentisierung sowie Zertifikate einzusetzen.

[32] Bei fest vorgegebenen Algorithmen gilt es die Auswahl zu überschreiben, um durch Updates hinzugekommene Algorithmen nicht automatisch zu aktivieren

[33] Wird unter `KexAlgorithms` ein `diffie-hellman-group-exchange`-Algorithmus gewählt, sind die Gruppen anhand der Option `ModuliFile` und entsprechender Datei (Standard `/etc/moduli`) einzuschränken [3]

3.5.4.3. Forwarding / Weiterleitungen

Das Forwarding des Authentisierungs-Agenten sowie von Unix Sockets und TCP-Ports via Local oder Remote Forwarding ist standardmässig erlaubt [3]. Der offene Port für das Local oder Remote Forwarding ist jedoch so eingeschränkt, dass diese nur über die lokale Loopback-Adresse und somit nicht von anderen Hosts erreichbar ist (Option `GatewayPorts`) [3].

Dies erlaubt es zum Beispiel auf einem Client ohne Internet mittels SSH-Server, welcher über einen Internetzugang verfügt, eine Internetverbindung durch Local Forwarding herzustellen. Der SSH-Client-Befehl dazu würde wie folgt lauten, um über den Server auf „www.openbsd.com“ unter Port 443 über dessen lokalen Port 11443 zugreifen zu können [34]:
`ssh -L 11443:www.openbsd.com:443 user@server`.

Mit Ausgabe von Debugging-Meldungen mit `-v` (siehe Kapitel 3.4.1.1) erscheint auf dem Client nach dem Login folgende Meldung:

```
1 debug1: Local connections to LOCALHOST:11443 forwarded to remote address www.
     openbsd.com:443
2 debug1: Local forwarding listening on ::1 port 11443.
3 debug1: channel 0: new port-listener [port listener] (inactive timeout: 0)
4 debug1: Local forwarding listening on 127.0.0.1 port 11443.
5 debug1: channel 1: new port-listener [port listener] (inactive timeout: 0)
```

Quelltext 3.8: Ausgabe des SSH-Clients bei Local Forwarding über Port 11443 zu www.openbsd.com:443

Der Aufruf via Webbrowser auf dem Client zur Loopback-Adresse (in diesem Falle die IPv6-Adresse „::1“) sieht dann wie folgt aus:

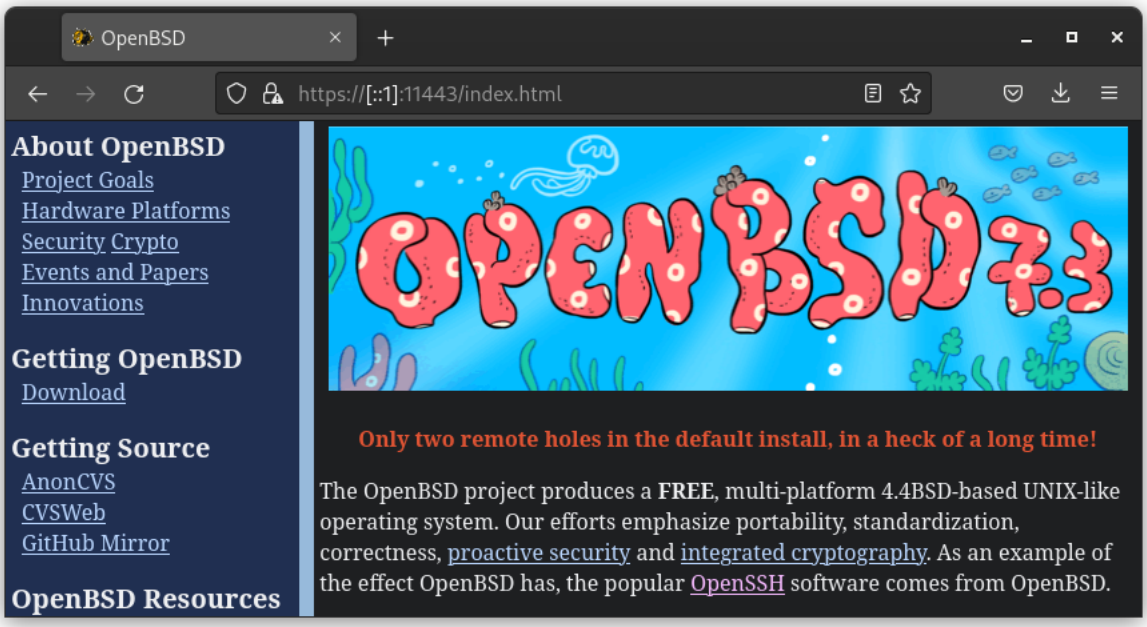

Abbildung 3.3.: Aufruf von „https://[::1]:11443“, zeigt via Local Forwarding über einen SSH-Server zu „www.openbsd.com:443“

[34] Je nach internen Weiterleitungen auf den Webseiten kann die Webseite nicht verfügbar sein, aber für einfache Webseiten wie „www.openbsd.com“ funktioniert es („www.openssh.com“ würde aber schon auf die Webseite von „www.openbsd.com“ weiterleiten beim Aufruf ohne Domainnamen), wobei alternativ anderer Verkehr als HTTP(S) weitergeleitet werden kann

Daher sind Weiterleitungen über TCP-Ports und Unix Sockets über die entsprechenden Optionen (`AllowTcpForwarding` und `AllowStreamLocalForwarding`) zu unterbinden. Sollte die Funktionalität doch benötigt werden, gilt es mit den `Permit`-Optionen erlaubte Ports und Ziel-Adressen zu hinterlegen.

Ob der Authentisierungs-Agent eines Clients auf dem Server für weitere Verbindungen davon weiterverwendet werden darf, ist ebenfalls mittels `AllowAgentForwarding` zu definieren. Ein OpenSSH-Entwickler bewertet das Agent-Forwarding als „riskante Operation“ und empfiehlt stattdessen die `ProxyJump`-Option (`-J`) des SSH-Clients (siehe Kapitel 3.4.1.1) [35] [96]. Sofern kein Anwendungsfall existiert, bei welchem von einem Server auf einen anderen verbunden werden muss, gilt es das Agent-Forwarding zu deaktivieren.

3.5.4.4. Spezifische Konfiguration

Die OpenSSH-Server-Konfiguration wird nur mit allgemein gültigen Optionen ausgeliefert [3]. Sollte unter einer bestimmten Bedingung anhand eines u.a. gegebenen Public-Keys, bestimmten Benutzers oder Hosts eine Funktionalität gestattet werden, die normalerweise deaktiviert ist, so ist diese auf die Bedingung spezifisch einzuschränken (z.B. mit `Match`-Parameter oder Public-Key-Optionen).

Nur weil ein bestimmter Benutzer z.B. TCP-Forwarding benötigt, ist es nicht global zu aktivieren.

[35] Diese Empfehlung ist ebenfalls in der SSH-Client-Manpage [47] vorzufinden

3.6. Schwachstellen bzw. Verwundbarkeiten

Dieses Kapitel zeigt einen Überblick zu bekannten Schwachstellen und Software-Fehlern („Bugs“) von OpenSSH und deren Umgang von dessen Entwicklern. Die OpenSSH-Entwickler publizieren auf ihrer Webseite in den Bereichen „Release Notes“ [63] und „Security“ [97] entsprechende Anpassungen an der Software sowie bekannte Verwundbarkeiten.

Öffentliche Schwachstellen betreffend Cybersecurity werden mit dem CVE-Programm mit einer CVE-ID sowie gegebenenfalls mit CVSS-Punkten zur Bewertung des Schweregrades von 0.0 bis 10.0 publiziert [98][99]. Sämtliche CVE-Einträge können als täglich aktualisierte CVE-Liste heruntergeladen und analysiert werden [100]. Zur Übersicht können entsprechende Schachstellen auf der CVE-Webseite anhand der CVE-ID eingesehen werden [101].

Im Rahmen dieser Arbeit wurde die CVE-Liste am 29. Juli 2023 als Archiv heruntergeladen [100], extrahiert und mit folgendem Befehl durchsucht: `grep -lri "\"value\".*openssh" | sort` [36]. Daraus erfolgt eine Liste an CVE-Einträgen, wessen Beschreibung OpenSSH beinhalten, wobei sich nicht jeder Eintrag direkt mit der OpenSSH-Implementation befasst, sondern auch Produkte behandelt werden, wessen Verwendung von OpenSSH in einer bestimmten Konfiguration eine Schwachstelle verursacht (z.B. verwendet bei CVE-2020-5917 ein Produkt einen OpenSSH-Server mit Schlüsseln, die kleiner als 2048 Bit sind [102]).

Der Vergleich zwischen der durchsuchten CVE-Liste und den Informationen der OpenSSH-Entwickler auf deren Security-Webseite [97] zeigt auf, dass die OpenSSH-Security-Webseite lediglich zu ihrer Implementation anerkannte Schwachstellen aufweist und die CVE-Einträge auch umstrittene Verhalten beschreiben und auf entsprechende Diskussionen verweisen, z.B. CVE-2021-36368 [103] [37]. Des Weiteren werden auf der OpenSSH-Webseite nur selten zugehörige CVE-Einträge aufgeführt, wobei entsprechende Schwachstellen ohne CVE-ID auf der Webseite anhand deren Beschreibung ausfindig gemacht werden können (z.B. CVE-2023-28531 [104], wessen ID auf der OpenSSH-Webseite nicht aufgefunden werden kann). Die Release Notes von OpenSSH [63] behandeln auch CVE-Einträge, welche nicht auf der Security-Seite [97] aufgeführt werden (z.B. CVE-2019-6111).

Daraus entsteht der Eindruck, dass für eine genauere Übersicht die CVE-Liste zu konsultieren ist, während deren Behandlungen in verlinkten Diskussionen, der OpenSSH-Webseite [46] oder anderen Webseiten nachgeschaut werden muss. Im optimalen Fall verweist die OpenSSH-Webseite [46] auf entsprechende CVE-IDs.

Zur Beurteilung der Reaktionszeit der OpenSSH-Entwickler wurden CVE-Einträge der letzten 5 Jahre (ab 2018 bis und mit 29. Juli 2023) zu OpenSSH gemäss vorher aufgeführtem Vorgehen ausgelesen und nicht-relevante Einträge manuell herausgefiltert. Daraufhin wurde jeder CVE-Eintrag anhand der dort verlinkten Informationen geprüft, ob und wann dieser behandelt wurde. Das Ergebnis ist der nächsten Tabelle zu entnehmen.

[36] Mit diesem Befehl werden die Dateien im Ordner rekursiv (`-r`) und mit Ignorieren der Gross- und Kleinschreibung (`-i`) nach Zeilen mit dem Inhalt `value` darauffolgendem Text `openssh` durchsucht und die zutreffenden Dateinamen angezeigt (`-l`) sowie deren Ergebnis sortiert ausgegeben (`| sort`)

[37] Bei diesem Fall meldet ein OpenSSH-Entwickler „this is not an authentication bypass, since nothing is being bypassed“, wobei der Fall über ein halbes Jahr später durch ein neues Feature gelöst werden konnte [96]

CVE-ID CVSS-3.1-Score CWE (Common Weakness Enumeration)	Datum Publikation	OpenSSH-Version mit Behebung
CVE-2023-38408 [105] Noch kein Score vergeben Kein CWE zugewiesen [106]	19.07.2023	9.3p2 19.07.2023 [63]
CVE-2023-28531 [104] 9.8 Kein CWE zugewiesen [107]	17.03.2023	9.3/9.3p1 15.03.2023 [63]
CVE-2023-25136 [108] 6.5 CWE-415 Double Free [109]	03.02.2023	9.2/9.2p1 02.02.2023 [63]
CVE-2021-41617 [110] 7.0 Kein CWE zugewiesen [111]	26.09.2021	8.8/8.8p1 26.09.2021 [63]
CVE-2021-36368 [112] 3.7 CWE-287 Improper Authentication [113]	12.03.2022	8.9/8.9p1 23.02.2022 [63] [96]
CVE-2021-28041 [114] 8.5 CWE-415 Double Free [115]	05.03.2021	8.5/8.5p1 03.03.2021 [63]
CVE-2020-15778 [116] 7.8 CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') [117]	24.07.2020	8.4/8.4p1 27.09.2020 [63][118]
CVE-2020-14145 [119] 5.9 CWE-203 Observable Discrepancy [120]	29.06.2020	8.7/8.7p1 20.08.2021 [121]
CVE-2020-12062 [122] 7.5 CWE-20 Improper Input Validation [123]	01.06.2020	8.3/8.3p1 27.05.2020 [63]
CVE-2019-16905 [124] 7.8 CWE-190 Integer Overflow or Wraparound [125]	09.10.2019	8.1/8.1p1 09.10.2019 [63]
CVE-2019-6111 [126] 5.9 CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') [127]	31.01.2019	8.0/8.0p1 17.04.2019 [63]
CVE-2019-6110 [128] 6.8 CWE-838 Inappropriate Encoding for Output Context [129]	31.01.2019	8.0/8.0p1 17.04.2019 [63]
CVE-2019-6109 [130] 6.8 CWE-116 Improper Encoding or Escaping of Output [131]	31.01.2019	8.0/8.0p1 17.04.2019 [63]
CVE-2018-20685 [132] 5.3 CWE-863 Incorrect Authorization [133]	10.01.2019	8.0/8.0p1 17.04.2019 [63]
CVE-2018-15919 [134] 5.3 CWE-200 Exposure of Sensitive Information to an Unauthorized Actor [135]	28.08.2018	7.9/7.9p1 19.10.2018 [63]
CVE-2018-15473 [136] 5.3 CWE-362 Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') [137]	17.08.2018	7.8/7.8p1 24.08.2018 [63]

Tabelle 3.5.: OpenSSH-CVE-Einträge ab 2018 bis und mit 29. Juli 2023

Die genauere Definition eines CWE-Typen kann der CWE-Webseite [138] entnommen werden.

Aus der Tabelle lässt sich entnehmen, dass publizierte Schwachstellen zu OpenSSH meist vor oder am Tag der CVE-Publikation durch eine neue Version behoben wurden. Je nach Schweregrad bzw. CVSS-Score kann es auch vorkommen, dass diese erst einige Monate später behoben wurden. Weiterhin wird damit hervorgehoben, dass es wichtig ist, die Software laufend auf dem aktuellsten Stand zu halten und nur nötige Funktionalitäten zu aktivieren, um eine möglichst kleine Angriffsfläche zu bieten. Aktuell sind bezüglich OpenSSH keine offenen Schwachstellen bekannt, jedoch kann jederzeit eine publiziert werden [100].

Wie bereits erwähnt, erfolgt auf der OpenSSH-Webseite kaum eine Verknüpfung zu den CVE-Einträgen. Daher wird empfohlen eine sich Schwachstellen-Übersicht anhand der CVE-Liste [100][101] zu verschaffen und Auseinandersetzungen jeweils entsprechender Links der CVE-Einträge zu entnehmen. Die Webseite von OpenSSH zeigt im Security-Bereich [97] einen groben Überblick zu dessen Schwachstellen auf und geht in den entsprechenden Release Notes [63] auf die entsprechende Schwachstelle und dessen Behebungen ein (auch zu welchen, die nicht im Security-Abschnitt aufgeführt sind oder vorher entdeckt werden).

Dadurch, dass der Quellcode von OpenSSH innerhalb des Code-Versionierungs-Systems von OpenBSD [139] eingesehen werden kann, ist es möglich auf entsprechende Anpassungen zu verlinken und Änderungen direkt nachzuvollziehen. Wird OpenSSH auf OpenBSD betrieben, kann das ganze Betriebssystem inklusive OpenSSH mittels `syspatch`-Befehl aktualisiert werden [38] [141].

OpenSSH verwendet für einige kryptographische Operationen die Programmbibliothek LibreSSL [13]. Diese und weitere Bestandteile des Betriebssystems sowie darunterliegende Hardware können ebenfalls Verwundbarkeiten aufweisen und Einfallstore für Angriffe darstellen. Für eine vollumfängliche Überprüfung gilt es ebenfalls diese Komponenten zu prüfen und die Angriffsfläche möglichst klein zu halten.

Der zuvor erwähnte Befehl `syspatch` in OpenBSD umfasst zwar das ganze Betriebssystem und enthält auch Security Updates [142], jedoch sind zusätzlich installierte Anwendungen separat zu pflegen. Mit `pkg_add` installierte Pakete können in OpenBSD mit `pkg_add -u` aktualisiert werden [143].

[38] Angewendete Patches werden unter OpenBSD im Verzeichnis `/var/syspatch` abgelegt und können bei Bedarf auch wieder zurückgesetzt werden [140]. Als Beispiel ist der OpenSSH-Patch vom 19.07.2023 bei OpenBSD 7.3 nach Ausführen von `syspatch` unter `/var/syspatch/73-010_ssh_agent/010_ssh_agent.patch.sig` einlesbar, wobei hier nur `ssh-agent` betroffen war

3.7. Interpretation

Betrachtet man die Algorithmen-Wahl des SSH-Servers `sshd` anhand der Standardeinstellungen dessen Konfiguration (siehe Abschnitt „Algorithmen-Wahl“ in Kapitel 3.4.3.1), ergibt sich die Feststellung, dass sich das OpenSSH-Projekt sehr nahe am Stand der Technik sowie verfügbaren Publikationen befindet, was sich auch in dessen Software-Release-Notes [63] spiegelt.

OpenSSH stellt für dessen symmetrische Verschlüsselungen den Algorithmus „ChaCha20-Poly1305“ mit höchster Priorität voran [3], welcher in keinem der betrachteten Dokumente einer Behörde erwähnt wurde. Der Schlüsselaustausch-Algorithmus „Streamlined NTRU Prime“ wird von OpenSSH als primärer Algorithmus hinterlegt [3]. Beide erwähnten Algorithmen werden in den betrachteten Empfehlungs-Publikationen der Bundesbehörden nicht erwähnt, wobei letzterer Algorithmus nach dessen Einbindung in OpenSSH aus dem „Post-Quantum Cryptography Standardization Project“ der NIST ausgeschieden ist [144]. Daraus lässt sich schliessen, dass bezüglich Quantencomputern noch eine gewisse Unklarheit besteht und sich in Zukunft noch einiges ändern kann.

Würde die Anforderung bestehen, z.B. nur NIST-bewilligte Algorithmen zu nutzen, wäre die SSH-Server-Konfiguration entsprechend anzupassen. Vertraut man hingegen der Algorithmen-Wahl von OpenSSH, so wird diese im Laufe der Zeit durch neue Versionen angepasst [63]. Bei fest hinterlegten Algorithmen ist die Auswahl regelmässig zu prüfen und ggf. zu aktualisieren.

Die standardmässig aktivierte Passwort-Authentisierung erlaubt es eine einfachere Initial-Installation durchzuführen, bei welchen neue Server meist zu Beginn nur einen Benutzernamen und Kennwort hinterlegt haben und Schlüssel erst später dazu kommen. Wird hier ein schlechtes Passwort gewählt und die Konfiguration des SSH-Servers nicht weiter angepasst entsteht ein Sicherheitsrisiko. Weiterhin kann die Verwendung von Zertifikaten ausgestellt durch eine CA (Certificate Authority) mehr Kontrolle über zugelassene Schlüssel bringen und somit die Sicherheit erhöhen, wobei dies einen höheren Aufwand mit sich bringt.

Die Wahrscheinlichkeit einer DoS Attacke wird durch das festlegen einer maximalen Anzahl an nicht-authentisierten Verbindungen minimiert (siehe Abschnitt „Maximalwerte und Timing-spezifisches“ in Kapitel 3.4.3.1).

Bezüglich Authentisierungs-Agent ratet die Manpage zum SSH-Client von dem Gebrauch dessen Weiterleitungsfunktionalität ab und schlägt die Verwendung eines Jump Hosts mit Parameter `-J` vor [47]. Beide Fälle werden durch die Standardkonfiguration initial ermöglicht [3].

Die Standardeinstellungen der SSH-Server-Konfiguration bieten somit einen Kompromiss zwischen Sicherheit und Komfort. Die Anwendung kann in ihrem Auslieferungszustand direkt gestartet und verwendet werden, womit ein Secure Channel etabliert wird, welcher gemäss dem aktuellen Stand der Technik einigermassen abgesichert ist. Die aktivierten Authentisierungsmethoden und Weiterleitungsoptionen erlauben eine simple Passwort-Authentisierung für alle Benutzer ausser `root` sowie Local oder Remote Forwarding mit Anbindung an eine lokale Loopback-Adresse. Zur weiteren Absicherung und Kontrolle gilt es somit weitere Anpassungen gemäss dem ermittelten Minimalstandard aus Kapitel 3.5.3 und dessen Vergleich aus Kapitel 3.5.4 vorzunehmen, um nur vorgesehene Datenflüsse zu erlauben und eine Übersicht über erlaubte Aktivitäten zu haben. Eine zentrale Überwachung entsprechender Logs ist ebenfalls zu empfehlen, um abweichende Verhalten festzustellen.

4. Aufbau

4.1. Laborumgebung

Die Konfiguration und Verifikation von OpenSSH findet in einer eigens aufgebauten Laborumgebung statt. Dazu werden auf einem Laptop drei virtuelle Maschinen in VirtualBox mit OpenBSD sowie eine virtuelle Maschine mit pfSense als Firewall/Router in Versionen gemäss Tabelle 2.1 installiert.

Hostname	Beschreibung	IPv4-Adresse(n)
c1.lab.internal	Client	192.168.100.1/24
s1.lab.internal	Server	192.168.1.1
s2.lab.internal	Server	192.168.2.1/24
fw.lab.internal	Firewall	192.168.100.254/24 192.168.1.254/24 192.168.2.254/24 NAT-Client-IP Internet (DHCP)

Tabelle 4.1.: Virtuelle Maschinen in der Laborumgebung

Die VMs befinden sich zusammen in Netzwerken, welche über die Firewall verbunden werden. IP-Routing ist bei OpenBSD standardmässig deaktiviert [141]. Folgende Datenflüsse sind zugelassen:

Quelle	Ziel	Port
c1	s1	tcp/22 SSH-Server-Standardeinstellung bei `Port`
s1	s2	tcp/22
c1, s1, s2	Internet bzw. alles ausser den anderen internen Netzwerken	tcp/80 (HTTP), tcp/443 (HTTPS), udp/123 (NTP) Webseitenzugriff mit HTTP(S) und Zeitsynchronisation mit NTP

Tabelle 4.2.: Datenflüsse in der Laborumgebung

Daraus ergibt sich folgende Abbildung zur Darstellung der Laborumgebung:

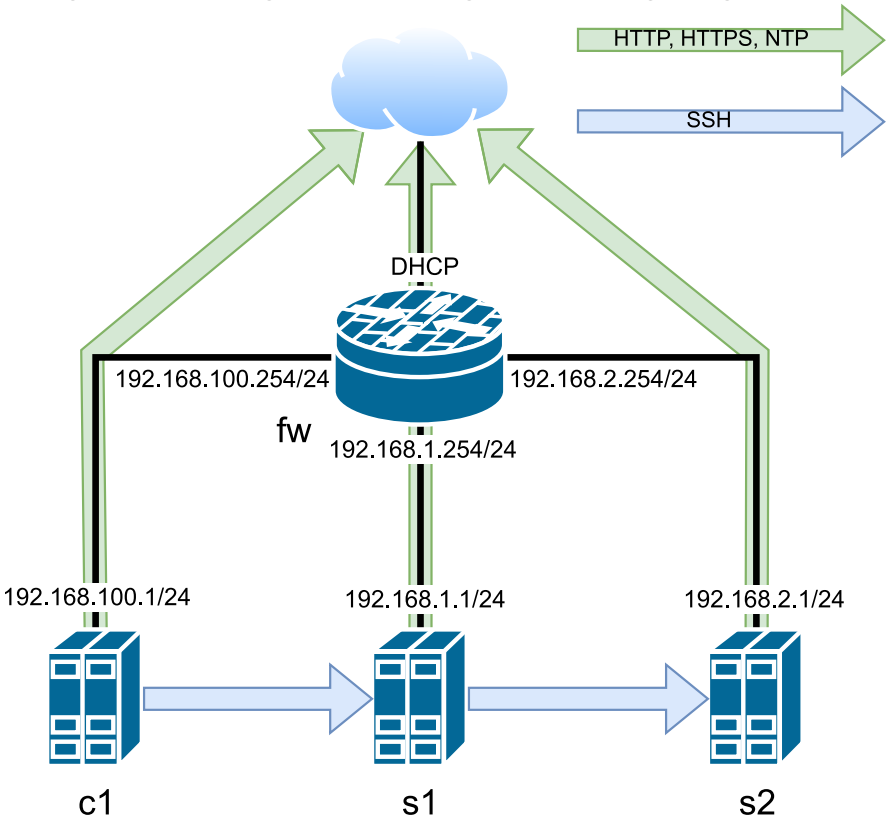

Abbildung 4.1.: Laborumgebung mit IP-Adressen und zugelassenen Datenflüssen

Jede VM wird mit 1 CPU, 1024 MB Arbeitsspeicher und 16 GB Speicher ausgestattet. Jeder VM ausser der Firewall wird ein Netzwerkadapter mit eigenem Netzwerk zugewiesen und dort mit der Firewall verbunden, um Installationsdateien und ggf. Patches herunterladen zu können.

4.1.1. Firewall-VM

Die Installation der Firewall erfolgt mit den Standardwerten des pfSense-Installationsassistenten, worauf die Netzwerkinterfaces entsprechend Tabelle 4.1 konfiguriert werden [39] und das WebGUI in VirtualBox mittels Port-Forwarding auf dem Host-Rechner zur Verfügung gestellt wird. Dann werden über das WebGUI Hostname, Domain, DNS-Server und Firewall-Rules festgelegt sowie beim WAN-Interface private IP-Adressen nicht blockiert. Als Platz- und Zeitgründen wird nicht weiter auf den Aufbau der Firewall eingegangen, da diese auch problemlos durch eine andere Firewall oder das Anbinden der VMs in mehreren Netzen ersetzt werden kann. Das Hauptziel ist hier sicherzustellen, dass `c1` und `s2` sich nicht direkt sehen können und zukünftig über `s1` verbunden werden müssen. Der Einsatz der Firewall erlaubt eine komfortablere Aktualisierung und ggf. Erweiterung der Laborumgebung. Zur Übersicht folgt eine Ansammlung an Screenshots der Firewall-Konfiguration:

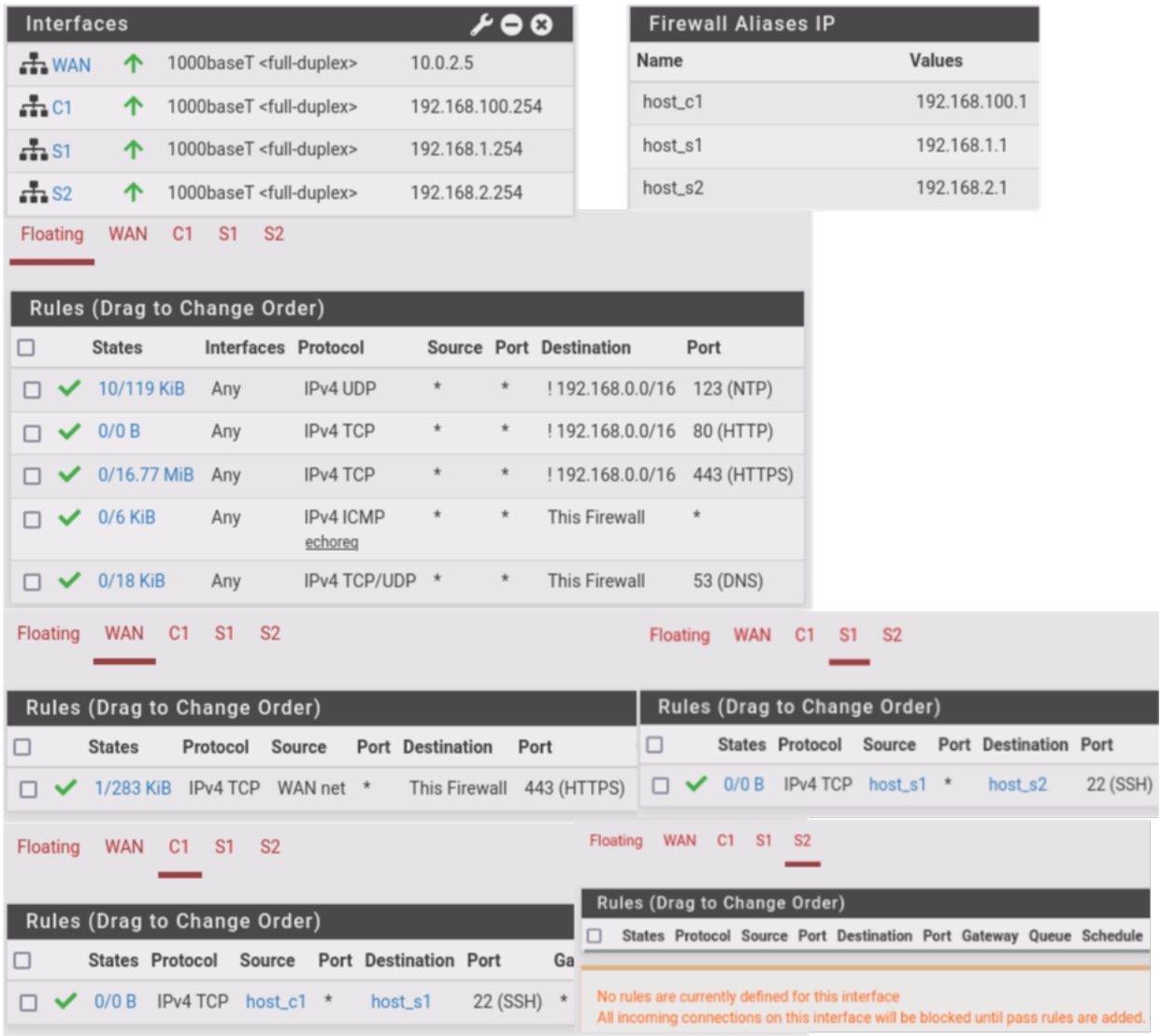

Abbildung 4.2.: Konfiguration der Firewall in der Laborumgebung

[39] Für zwei Interfaces (`em2` und `em3`) musste in der Shell via Konsole der Befehl `ifconfig emX up` ausgeführt werden, bevor das Interface zur Auswahl bereitstand und mit dem Assistent in der Konsole konfiguriert werden konnte

4.1.2. Client- und Server-Betriebssystem

Das Feature „I/O APIC“ ([145]) muss für OpenBSD in der gegebenen Version in den Motherboard-Eigenschaften der VM in VirtualBox deaktiviert werden, sonst bricht der Installationsvorgang mit einer „Panic“-Meldung ab [146].

Die OpenBSD-Installation erfolgt ab der ISO-Datei der OpenBSD Webseite [5]. Mit dem Installationsassistenten (Option `I` nach Booten ab ISO-Datei) werden folgende Eigenschaften in aufgeführter Reihenfolge festgelegt:

- Tastaturlayout `sg` (Swiss German)
- Hostname in Kurzform gemäss Tabelle 4.1 (ohne Domäne)
- IP-Adresse für Interface `em0` gemäss Tabelle 4.1 (nur IPv4)
- DNS Domain Name gemäss Tabelle 4.1 ohne Nameserver
- Passwort für `root`-Konto
- `sshd` nicht automatisch starten
- Frage betreffend Verwendung des „X Window Systems“ negativ beantworten
- Benutzer `user` inkl. Passwort anlegen
- Zeitzone `Europe/Zurich` wählen
- Disk gemäss Standardwerten (automatische Partitionierung, keine Verschlüsselung)
- Als Quelle für die Sets `http` und dann einen Server aus der Liste wählen (ohne Proxy)
- Sets abwählen: `-game* -x*` (Spiele und GUI-Komponenten [147])
- Verbleibende Sets herunterladen, verifizieren und installieren
- ISO-Datei trennen und System neu starten
- Als `root` einloggen
- Der während der Installation erstellte Benutzer wird automatisch zur Gruppe `wheel` hinzugefügt [141], worauf nach Konfiguration von `doas` Benutzer dieser Gruppe Befehle mit `doas` als `root` ausführen dürfen:
  `cp /etc/examples/doas.conf /etc/`
  Fortan kann ein Benutzer der Gruppe `wheel` mit `doas BEFEHL` ein Befehl als `root` ausführen, ähnlich zum Befehl `sudo` unter Linux-Betriebssystemen [40]
- Datei `/etc/mygate` erstellen mit IP-Adresse der Firewall im zugehörigen Netz (für `c1` also `192.168.100.254`), um den Default Gateway zu hinterlegen [148]
- Datei `/etc/resolv.conf` so anpassen, dass die `nameserver`-Zeile auf die IP-Adresse der Firewall im zugehörigen Netz zeigt (für `c1` also `nameserver 192.168.100.254`) [148]
- Mit `sh /etc/netstart` die Netzwerk-Einstellungen übernehmen [148]
- Security Updates und sonstige Reparaturen/„Fixes“ mit `syspatch` installieren [142]
- Pakete mit `pkg_add -u` aktualisieren, wobei nach einer neuen Installation nur das „quirks“-Paket mit Informationen zu anderen Paketen vorhanden ist [149][143]
- System mit `reboot` neu starten, um ggf. den neuen Kernel nach dem Update zu laden

[40] Es wird empfohlen `doas` zu verwenden und so wenig wie möglich direkt als `root` zu arbeiten, um nur das Nötigste als `root` auszuführen. Mit `doas` werden die ausgeführten Befehle zur Kontrolle zusätzlich unter `/var/log/secure` geloggt [142]

4.2. Einrichtung von OpenSSH

Zu Beginn wird für simple Infrastrukturen eine möglichst sichere Konfiguration erstellt, wobei im späteren Verlauf Zusatzoptionen zur weiteren Absicherung wie Zertifikate und CAs in Kapitel 4.2.8 sowie YubiKeys in Kapitel 4.2.11 behandelt werden.

Für jeden in der Zielsetzung (siehe Kapitel 1.2) definierten Anwendungsfall wird ein dedizierter Benutzer angelegt, um die Konfigurationen entsprechend aufteilen zu können. Dazu werden auf jedem Server (`s1` und `s2`) mittels Befehl `adduser` folgende Benutzer hinzugefügt und dabei die Standardoptionen gewählt:

- Kommandozeilenzugriff: `cmd`
- Dateiübertragungen: `file`
- Jumphost: `jump`
- Agent-Forwarding: `agent`

Jeder dieser 4 Benutzer wird auf den Servern der Gruppe `sshaccess` (hinzugefügt mittels `groupadd sshaccess`) zusätzlich mit folgendem Befehl zugewiesen: `usermod -G sshaccess USER`. Anhand dieser Gruppe soll zusätzlich der SSH-Zugriff eingeschränkt werden.

Die Ausführung des SSH-Clients erfolgt auf dem Client-Host `c1` jeweils unter dem Benutzer `user`.

Die Konfiguration orientiert sich am ermittelten Minimalstandard aus Kapitel 3.5.3 und dem daraus erfolgten Vergleich mit den SSH-Server-Standardeinstellungen aus Kapitel 3.5.4. Stehen mehrere Optionen zur Verfügung, wird darauf hingewiesen.

4.2.1. Erstellen von Schlüsseln zur Public-Key-Authentisierung

Der Benutzer `user` auf dem Client-Host `c1` verfügt zu Beginn über keine eigenen Private-Keys, welche er für die Public-Key-Authentisierung verwenden könnte, welche gemäss dem ermittelten Minimalstandard aus Kapitel 3.5.3 zu aktivieren ist.

Wie folgt wird mit `ssh-keygen` wird ein Schlüssel des Typs Ed25519 (EdDSA mit „Safe Curve“ Curve25519) erstellt:

```
c1$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/user/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_ed25519
Your public key has been saved in /home/user/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk user@c1.lab.internal
The key's randomart image is:
+--[ED25519 256]--+
|       .o*+..     |
|        = .E  .   |
|       o .. o+ .  |
|        . .+o.+ o.|
|         S .+.o=.o|
|            .+oooo|
|             .@oB.|
|            .+o&o+|
|             .+=++|
+----[SHA256]-----+
```

Quelltext 4.1: Erstellung eines Ed25519-Schlüssels mit `ssh-keygen` unter `~/.ssh/id_ed25519`

Der Standardpfad für den neuen Schlüssel lautet `/home/user/.ssh/id_ed25519`, kann im Dialog jedoch angepasst oder mit dem Parameter `-f` mitgegeben werden. Eine Passphrase ist gemäss Minimalstandard aus Kapitel 3.5.3 zu definieren. Der zugehörige Public-Key wird unter `/home/user/.ssh/id_ed25519.pub` abgelegt. Als Standardkommentar wird der Benutzername und der Hostname mit dem „@"-Symbol verknüpft verwendet, wobei dieser mit der Option `-C` anders definiert werden könnte.

Gemäss Minimalstandard (Kapitel 3.5.3) sind zusätzlich folgende Optionen für die Schlüsselerstellung möglich:

- ECDSA mit NIST-Kurve P-256 `-t ecdsa -b 256`, P-384 `-t ecdsa -b 384` oder P-521 `-t ecdsa -b 521`
- RSA mit 3072 Bit Schlüssellänge: `-t rsa -b 3072`

Die Dateinamen der Schlüssel würden entsprechend dem gewählten Algorithmus, also `id_ed25519`, `id_ecdsa` oder `id_rsa` lauten.

Der Inhalt des Public-Keys (hier unter `/home/user/.ssh/id_ed25519.pub`) sieht wie folgt aus:

```
1 ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIMH4AMuabHjbA5M8rXbOY2EEy0Da/
     ZOtM4psNvSO50pc user@c1.lab.internal
```

Quelltext 4.2: Inhalt des Ed25519-Public-Keys unter `~/.ssh/id_ed25519.pub`

Es ist der Schlüsseltyp, der Public-Key selber und der zugehörige Kommentar ersichtlich. Um diesen Schlüssel für den Login auf einen Server mit Public-Key-Authentisierung verwenden zu können, muss der gezeigte Inhalt in das entsprechende `AuthorizedKeysFile` (siehe Abschnitt „Authentisierung und Autorisierung: Public-Key-Authentisierung" in Kapitel 3.4.3.1) gepflegt werden.

Damit der Benutzer `user` auf dem Client-Host `c1` nun mit dem Benutzer `cmd` auf dem Server-Host `s1` einloggen kann, muss der Public-Key von `user@c1` nach `cmd@s1`.

Der Inhalt von `/home/user/.ssh/id_ed25519.pub` auf `c1` muss somit manuell an die Datei `/home/cmd/.ssh/authorized_keys` auf `s1` angefügt werden. Sofern der Login mit lediglich einem Passwort noch erlaubt ist, kann dies mittels folgendem Befehl gelöst werden [41]:

```
1 cat /home/user/.ssh/id\_ed25519.pub | ssh cmd@192.168.1.1 'cat >> .ssh/
     authorized\_keys'
```

Quelltext 4.3: Anfügen von `~/.ssh/id_ed25519.pub` auf `c1` an `/home/cmd/.ssh/authorized_keys` auf `s1`

Alternativ kann dies mittels Zusatztool `ssh-copy-id` [42] als `user` auf `c1` ebenfalls ausgeführt werden: `scp-copy-id cmd@192.168.1.1`. Für beide dieser Befehle muss man den Login für den Benutzer (hier `cmd`) auf dem Zielhost (hier `192.168.1.1`) kennen und der SSH-Serverdienst auf dem Zielhost bereits mindestens in der Standardkonfiguration gestartet sein (siehe Kapitel 4.2.4).

Nun ist der Public-Key von `user@c1` bei `cmd@s1` hinterlegt. Beim nächsten Login mittels `ssh cmd@192.168.1.1` wird nach der Passphrase des erstellten Private-Keys von Benutzer `user` gefragt, nicht mehr nach dem Passwort von Benutzer `cmd`. Somit erfolgt in diesem Fall nun eine Public-Key-Authentisierung.

Derselbe Public-Key wird bei den restlichen, in Kapitel 4.2 erstellten Benutzern (`file`, `jump`, `agent`) auf dieselbe Art wie bei Benutzer `cmd` auf dem Server-Host `s1` hinterlegt. Zusätzlich wird der Public-Key beim Benutzer `cmd` auf dem Server `s2` hinzugefügt.

[41] Hier wird der Public-Key mittels `cat` in die Standardausgabe ausgegeben, welche mittels `|` an den `ssh`-Befehl weitergeleitet wird, in welchem mittels eines weiteren `cat`-Befehls die weitergeleitete Ausgabe wieder ausgegeben und an die hinterlegte Datei angefügt wird

[42] Muss unter OpenBSD installiert werden mittels `pkg_add ssh-copy-id`

4.2.2. Grundkonfiguration

Die Grundkonfiguration beschreibt die allgemein gültigen Optionen für den OpenSSH-Serverdienst, welche für sämtliche folgenden Anwendungsfälle gelten, sofern diese dort nicht explizit überschrieben werden. Dazu wird die Standard-Konfigurationsdatei unter `/etc/ssh/sshd_config` angepasst. Gemäss Tabelle 3.3 in Kapitel 3.4.3.1 ist die Angabe alternativer Konfigurationsdateien möglich, jedoch eignet sich diese Option mehr zum Ausprobieren neuer Konfigurationen ohne die bestehende anzupassen, was hier nicht der Fall ist.

Gemäss dem ermittelten Minimalstandard aus Kapitel 3.5.3 und dem daraus erfolgten Vergleich mit den SSH-Server-Standardeinstellungen aus Kapitel 3.5.4 ergibt sich folgende Konfiguration für den OpenSSH-Serverdienst `sshd` in dessen Konfigurationsdatei `/etc/ssh/sshd_config` (Kommentare sind wo nötig mit „#"-Symbol auf englisch vor oder bei der entsprechenden Zeile hinterlegt) [43]:

```
# Listen ----------------------------------------------------------------
Port                            22
AddressFamily                   any
ListenAddress                   0.0.0.0
ListenAddress                   ::

# Private keys of server ------------------------------------------------
                                # Generated RSA server host key is 3072 bit
HostKey                         /etc/ssh/ssh_host_rsa_key
HostKey                         /etc/ssh/ssh_host_ecdsa_key
HostKey                         /etc/ssh/ssh_host_ed25519_key

# Algorithms ------------------------------------------------------------
# Used to fulfill the minimalstandard, ignore to trust selection of OpenSSH

CASignatureAlgorithms           ssh-ed25519,ecdsa-sha2-nistp256,ecdsa-sha2-
    nistp384,ecdsa-sha2-nistp521,sk-ssh-ed25519@openssh.com,sk-ecdsa-sha2-
    nistp256@openssh.com,rsa-sha2-512,rsa-sha2-256

                                # with AEAD, not just aes-cbc or aes-ctr
Ciphers                         aes128-gcm@openssh.com,aes256-gcm@openssh.com

HostKeyAlgorithms               ssh-ed25519-cert-v01@openssh.com,ecdsa-sha2-
    nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,
    ecdsa-sha2-nistp521-cert-v01@openssh.com,sk-ssh-ed25519-cert-v01@openssh.
    com,sk-ecdsa-sha2-nistp256-cert-v01@openssh.com,rsa-sha2-512-cert-
    v01@openssh.com,rsa-sha2-256-cert-v01@openssh.com,ssh-ed25519,ecdsa-sha2-
    nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,sk-ssh-ed25519@openssh.
    com,sk-ecdsa-sha2-nistp256@openssh.com,rsa-sha2-512,rsa-sha2-256

                                # "ssh -Q kex" shows no dh-group15 for DHKE
                                # but group18 with bigger size
KexAlgorithms                   diffie-hellman-group16-sha512,diffie-hellman-
    group18-sha512,curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-
    nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521

                                # Use HMAC
MACs                            hmac-sha2-256-etm@openssh.com,hmac-sha2-512-
    etm@openssh.com,hmac-sha2-256,hmac-sha2-512

RequiredRSASize                 3072

```

[43] Beschreibungen der Optionen sind in Abschnitt „SSH-Server Konfigurationsdatei sshd_config" in Kapitel 3.4.3.1 vorzufinden

```
32 PubkeyAcceptedAlgorithms      ssh-ed25519-cert-v01@openssh.com,ecdsa-sha2-
      nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,
      ecdsa-sha2-nistp521-cert-v01@openssh.com,sk-ssh-ed25519-cert-v01@openssh.
      com,sk-ecdsa-sha2-nistp256-cert-v01@openssh.com,rsa-sha2-512-cert-
      v01@openssh.com,rsa-sha2-256-cert-v01@openssh.com,ssh-ed25519,ecdsa-sha2-
      nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,sk-ssh-ed25519@openssh.
      com,sk-ecdsa-sha2-nistp256@openssh.com,rsa-sha2-512,rsa-sha2-256
33
34 # Ciphers and keying ------------------------------------------------------
35                             # Rekey after ciphers default amount,
36                             # no time based rekeying
37 RekeyLimit                  default none
38
39 # Logging  -----------------------------------------------------------------
40 SyslogFacility              AUTH
41 LogLevel                    INFO
42
43 # Authentication  ----------------------------------------------------------
44 LoginGraceTime              1m
45 StrictModes                 yes
46
47                             # Only allow pubkey and pw auth combined
48 AuthenticationMethods       publickey,password
49
50 PubkeyAuthentication        yes
51 AuthorizedKeysFile          .ssh/authorized_keys
52
53                             # Force PIN when using FIDO auth algo
54                             # (i.e. ecdsa-sk or ed25519-sk)
55 PubkeyAuthOptions           verify-required
56
57 HostbasedAuthentication     no
58
59 PasswordAuthentication      yes
60 PermitEmptyPasswords        no
61
62 KbdInteractiveAuthentication  no
63
64 # User / Group Filter ----------------------------------------------------
65 AllowGroups                 sshaccess
66 PermitRootLogin             no
67
68 # Forwarding / Tunnel ------------------------------------------------------
69 DisableForwarding           yes
70 AllowAgentForwarding        no
71 AllowStreamLocalForwarding  no
72 AllowTcpForwarding          no
73 PermitListen                none
74 PermitOpen                  none
75 GatewayPorts                no
76 X11Forwarding               no
77 PermitTunnel                no
78
```

```
79  # Other settings ------------------------------------------------------------
80                                  # Don't give user a terminal and exit session
81  PermitTTY                       no
82  ForceCommand                    exit
83
84  PrintMotd                       no
85  PrintLastLog                    no
86  TCPKeepAlive                    yes
87  PermitUserEnvironment           no
88  PermitUserRC                    no
89
90  Compression                     no
91
92  UseDNS                          no # activate if using DNS, not using DNS atm
93
94  PidFile                         /var/run/sshd.pid
95  MaxStartups                     10:30:100
96
97  ChrootDirectory                 none
98  VersionAddendum                 none
99  Banner                          none
100
101 # sftp subsystem
102 Subsystem                       sftp      /usr/libexec/sftp-server
```

Quelltext 4.4: SSH-Serverdienst-Grundkonfiguration `/etc/ssh/sshd_config` auf Server `s1` und `s2`

Die jeweils unterstützten Algorithmen konnten mittels `ssh -Q` ausgelesen werden, wobei der genaue Parameter der zugehörigen Manpage zu entnehmen ist [47].

Konkret wurden gegenüber der ausgelieferten Konfiguration folgende Anpassungen vorgenommen:

- `LoginGraceTime` von 2 Minuten auf 1 Minute gesetzt, um weniger gleichzeitige, nicht authentisierte Verbindungen offen zu haben
- `RequiredRSASize` von 1024 auf 3072 Bits festgelegt
- Den Login für Benutzer `root` mit `PermitRootLogin no` komplett deaktiviert
- „Keyboard-Interactive"-Authentisierung deaktiviert
- Ausgabe des letzten Login-Zeitstempels und „Message of the day" deaktiviert (`PrintLastLog no`, `PrintMotd no`)
- Allozieren eines Pseudo-Terminals unterbunden (`PermitTTY no`) und den Befehl `exit` forciert (`ForceCommand exit`)
- Deaktivierung des Benutzer-RC-Skripts `~/.ssh/rc` (siehe Liste in Kapitel 3.4.3.1, Punkt 5f)
- Deaktivierung der Kompression (`Compression no`), da diese laut Manpage von `ssh` [47] nur für langsame Verbindungen geeignet ist und die Kommunikation für schnelle Verbindungen verlangsamt
- Deaktivierung sämtlicher Forwarding-/Weiterleitungs-Optionen (`DisableForwarding yes`), wobei diese im einzelnen nochmals explizit deaktiviert wurden
- Definition der Algorithmen-Wahl gemäss ermitteltem Minimalstandard aus Kapitel 3.5.3, wobei die Diffie-Hellman-Gruppe `group15` nicht zur Auswahl stand, stattdessen die Gruppe `group18` mit grösserer Stärke [21] gewählt wurde
- Einschränkung des Logins auf Benutzer der Gruppe `sshaccess` (`AllowGroups sshaccess`)
- Forcierung der kombinierten Authentisierung mittels Public-Key und Passwort des Ziel-Benutzers (`AuthenticationMethods publickey,password`)
- Forcierung der Eingabe eines PIN-Codes bei Verwendung eines FIDO Authenticators mit Algorithmen `ecdsa-sk` oder `ed25519-sk` (`PubkeyAuthOptions verify-required`) [3]

Diese Konfiguration wird auf beiden Servern `s1` und `s2` unter `/etc/ssh/sshd_config` hinterlegt und damit der bestehende Inhalt überschrieben. Mit dieser Konfiguration ist es noch niemandem möglich via SSH auf dem Server zu arbeiten, da auch nach einem erfolgreichen Login mit dem forcierten Befehl `exit` in der Shell die Sitzung gleich wieder beendet wird. Hierfür wird nun der erste Anwendungsfall „Kommandozeilenzugriff" im nächsten Kapitel implementiert und danach der SSH-Serverdienst mit der neuen Konfiguration gestartet.

4.2.3. Kommandozeilenzugriff

Für den Kommandozeilenzugriff wird für den in Kapitel 4.2 eingerichteten Benutzer `cmd` eine spezifische Konfiguration hinterlegt, um entsprechenden Zugriff zu gewähren.

Dazu wird folgende Konfiguration an die bestehende Grundkonfiguration aus Kapitel 4.2.2 unter `/etc/ssh/sshd_config` auf Server `s1` angefügt:

```
# User cmd ------------------------------------------------------------------
Match User cmd
    PermitTTY                   yes
    ForceCommand                none
```

Quelltext 4.5: SSH-Serverdienst-Konfigurationszusatz für Kommandozeilenzugriff auf Server `s1`

Damit wird dem Benutzer `cmd` ein Pseudo-Terminal alloziert und kein Befehl forciert.

Wird diese Konfiguration zusammen mit der Grundkonfiguration aktiviert (siehe Kapitel 4.2.4), kann der Benutzer `user` auf dem Client-Host `c1` mittels Befehl `ssh cmd@192.168.1.1` per SSH einen Login mit dem Benutzer `cmd` auf dem Server-Host `s1` ausführen. Daraufhin wird dieser nach der Passphrase für seinen hinterlegten Schlüssel (siehe Kapitel 4.2.1) zur Public-Key-Authentisierung abgefragt und danach das Passwort für den Benutzer `cmd` angefordert. Kann beides erfüllt werden wird via SSH der Kommandozeilenzugriff als Benutzer `cmd` auf dem Server-Host `s1` ausgeführt.

Hierfür gilt es folgendes für den Benutzer `cmd` zu beachten:

- Der Benutzer kann mit dieser Konfiguration ebenfalls Dateiübertragungen mittels SFTP mit den Befehlen `scp` (siehe Kapitel 3.4.1.2) und `sftp` (siehe Kapitel 3.4.1.3) durchführen, jedoch nur mit Pfaden, unter welcher dieser berechtigt ist
  - Um dies zu unterbinden wäre die `Subsystem`-Definition für SFTP aus der Grundkonfiguration zu nehmen, jedoch lässt sich diese nicht in einem `Match`-Block unterbringen [44]
  - Gäbe es keinen Anwendungsfall für Dateiübertragungen könnte die `Subsystem`-Definition für SFTP komplett aus der Konfiguration gelassen werden, jedoch ist es dem Benutzer auch möglich über den `ssh`-Befehl Dateien zu übertragen (zum Beispiel mit Kombination des `cat`-Befehls gemäss Quelltext 4.3)

[44] Versucht man dies meldet der Test der SSH-Server-Konfiguration mit `/usr/sbin/sshd -t` folgendes:
`Directive 'Subsystem' is not allowed within a Match block`

- Es gilt den Benutzer auf dem Zielsystem entsprechend einzuschränken, da dieser mit `ForceCommand none` jeglichen Befehl ausführen oder Dateien auslesen kann, zu welchen er berechtigt ist und sich auch als anderen Benutzer anmelden kann, wenn er den Login dazu weiss [45]
  - Mittels `ForceCommand`-Parameter oder `command`-Parameter beim entsprechenden Public-Key im `AuthorizedKeysFile` kann ein Befehl hinterlegt werden, welcher nach dem Login ausgeführt wird und danach die Verbindung wieder trennt
  - Soll der Benutzer nur einen spezifischen Befehl ausführen können, kann dieser so mitgegeben werden
  - Bei mehreren Befehlen muss ein Skript geschrieben und angegeben werden, welches dem Benutzer eine fixe Auswahl bietet und seine Wahl einliest (z.B. mit `Enter "d" for "df -h", "t" for "top" and any other to exit` [46])

Das vorhin erwähnte Beispiel zur Benutzer-Einschränkung könnte so gelöst werden, indem man ein lokales Skript auf dem Server mit folgendem Inhalt unter z.B. `/home/cmd/selection.sh` anlegt und den `cmd`-Benutzer entsprechend berechtigt:

```
#!/bin/sh
echo 'Enter "d" for "df -h", "t" for "top" and any other to exit: '
read choice
case $choice in
        d)
                df -h
                ;;
        t)
                top
                ;;
        *)
                exit
                ;;
esac
```

Quelltext 4.6: Skript zur Einschränkung möglicher Befehle für den Gebrauch mit `ForceCommand`

Daraufhin ist die SSH-Server-Konfiguration für den Benutzer `cmd` wie folgt anzupassen:

```
# User cmd ------------------------------------------------------------------
Match User cmd
        PermitTTY                   yes
        ForceCommand                /home/cmd/selection.sh
```

Quelltext 4.7: SSH-Serverdienst-Konfiguration für Kommandozeilenzugriff mit `ForceCommand`-Beispiel angehängt an `/etc/ssh/sshd_config`

Nach Aktivierung dieser Konfiguration würde nach dem Login via SSH mit dem Benutzer `cmd` direkt das Skript gestartet werden und nach dessen Ausführung die Sitzung beenden. Dateiübertragungen via `scp` oder `sftp` würden damit ebenfalls nicht möglich sein.

Im Rahmen dieser Arbeit wird für den Benutzer `cmd` die Konfiguration mit `ForceCommand none` gewählt.

[45] Hat in OpenBSD die Gruppe mit ID 0 (`wheel`) Benutzer aufgelistet, dürfen nur diese Benutzer mit `root` einloggen [150]

[46] Mit `df -h` kann man den freien Disk-Speicher im „human-readable"-Format ausgeben und mit `top` Informationen über die Prozesse des Systems betrachten

4.2.4. SSH-Server Aktivierung und Konfigurationsanpassungen

Auf den Servern `s1` und `s2` wird mit dem Benutzer `user` eingeloggt und der SSH-**Serverdienst** `sshd` mit folgendem Befehl **aktiviert**, um beim nächsten Bootvorgang automatisch zu starten:
`doas rcctl enable sshd` [142]. Mit `rcctl ls on` können aktivierte Services aufgelistet werden, worunter nun `sshd` ebenfalls aufzufinden sein sollte.

Mit dem Befehl `doas rcctl start sshd` wird `sshd` **gestartet** und folgender Befehl ermöglicht es zu prüfen, ob der Service gestartet wurde: `doas rcctl check sshd`. Auf den Check-Befehl daraufhin sollte `sshd(ok)` ausgegeben werden, was auch automatisch beim Service-Start geschieht.

Wird die **Konfiguration** des SSH-Servers in der Datei `/etc/ssh/sshd_config` angepasst, so kann diese mittels `doas rcctl reload sshd` **neu eingelesen** werden. Bestehende Verbindungen werden hierbei nicht getrennt.

Vor der Konfigurationsübernahme empfiehlt es sich die neue Konfiguration mittels
`doas /usr/sbin/sshd -t` zu testen (siehe Tabelle 3.3 in Kapitel 3.4.3.1), worauf bei Fehlern eine entsprechende Ausgabe getätigt wird.

Zum Beispiel erscheint folgende Ausgabe bei einem Schreibfehler („Pubke**x**Authentication„ anstelle von „Pubke**y**Authentication“):

```
1 /etc/ssh/sshd_config: line 50: Bad configuration option: PubkexAuthentication
2 /etc/ssh/sshd_config: terminating, 1 bad configuration options
```

Quelltext 4.8: Ausgabe von `sshd`-Testmodus mit `-t` bei Schreibfehler in `PubkeyAuthentication`

Bei einer fehlerfreien Konfiguration wird der Test mit `-t` keine weitere Ausgabe tätigen.

Die getestete und effektive Konfiguration inklusive sämtlicher Standardeinstellungen kann mittels Parameter `-T` (siehe Tabelle 3.3 in Kapitel 3.4.3.1) ausgegeben werden. Diese Ausgabe eignet sich aufgrund ihres Ausgabeformats für den Vergleich mit Konfigurationen anderer Server.

Gemäss `syslog`-Konfigurationsdatei `/etc/syslog.conf` werden Meldungen der „Log-Facility“ `auth` und dem „Log-Level“ `info` nach `/var/log/authlog` geschrieben. Da dies ebenso den Werten von `sshd` entspricht (siehe Abschnitt „Logging“ in Kapitel 3.4.3.1), sind in dieser Datei Logs von `sshd` vorzufinden, welche als Benutzer `root` oder Mitglied der Gruppe `wheel` gelesen werden kann.

4.2.5. Dateiübertragungen

Für Dateiübertragungen wird für den in Kapitel 4.2 eingerichteten Benutzer `file` eine spezifische Konfiguration hinterlegt, um entsprechenden Zugriff zu gewähren. Mit folgender Konfiguration wird dem Benutzer der Gebrauch von SFTP forciert, wobei sich dieser nur innerhalb eines definierten Pfades bewegen kann und über keinen Kommandozeilenzugriff verfügt:

```
# User file ------------------------------------------------------------------
Match User file
    ForceCommand              internal-sftp
    ChrootDirectory           /data/sftp
```

Quelltext 4.9: SSH-Serverdienst-Konfigurationszusatz für Dateiübertragungen auf Server `s1`

Die Konfiguration wird der Datei `/etc/ssh/sshd_config` auf Server `s1` angehängt und gemäss Kapitel 4.2.4 übernommen.

Alle Komponenten des unter `ChrootDirectory` angegebenen Pfades müssen dem Benutzer `root` gehören und nicht von einem anderen Benutzer oder Gruppe beschrieben werden können [47] [3]. Nun sind innerhalb dieses Pfades Ordner zu erstellen, in welchen der Benutzer `file` ebenfalls über Schreibrechte verfügt. Die Umgebung wird demnach wie folgt aufgebaut:

```
s1$ doas mkdir -p /data/sftp/upload
s1$ doas mkdir /data/sftp/download
s1$ doas chown file /data/sftp/upload
s1$ ls -l /data/sftp/
total 8
drwxr-xr-x  2 root  wheel  512 Aug 12 20:52 download
drwxr-xr-x  2 file  wheel  512 Aug 12 20:52 upload
```

Quelltext 4.10: Aufbau Umgebung für SFTP-Dateiübertragungen mit Option `ChrootDirectory` auf Server `s1`

Somit existiert nun unter `/data/sftp/download` ein Pfad unter welchem der Benutzer `file` nur Daten lesen kann, wobei er unter `/data/sftp/upload` auch schreiben kann.

Der Pfad `/data/sftp` stellt für den Benutzer `file` via SFTP das Wurzel- und Heimverzeichnis dar. Soll demnach mittels `scp` eine Datei hochgeladen werden, ergeben in diesem Falle beide folgenden Beispiele dasselbe Ergebnis (Datei wird nach `/data/sftp/upload/hello.txt` auf den Zielhost `192.168.1.1` kopiert):

- `scp hello.txt file@192.168.1.1:upload/`
- `scp hello.txt file@192.168.1.1:/upload/`

Nach dem Login via `sftp` und Eingabe des Befehls `pwd` (zur Ausgabe des derzeitigen Arbeitsverzeichnisses) wird `Remote working directory: /` ausgegeben.

Somit können mit dem Benutzer `file` und dem Server `s1` mittels `scp` (siehe Kapitel 3.4.1.2) und `sftp` (siehe Kapitel 3.4.1.3) Dateiübertragungen von und zu dem Server durchgeführt werden, wobei die Befehle durch den Benutzer `user` auf dem Client-Host `c1` initiiert werden [48].

[47] Am einfachsten wird der Pfad als `root` mit `doas mkdir -p /data/sftp` erstellt
[48] Dies weil der Public-Key von `user` gemäss Kapitel 4.2.1 beim User `file` auf dem Server `s1` hinterlegt wurde

4.2.6. Jumphost / Zwischenrechner

In diesem Kapitel fungiert der Server `s1` als Zwischenrechner, um eine Verbindung zwischen dem Client `c1` und dem Server `s2` herstellen zu können. Gemäss Aufbau der Laborumgebung in Kapitel 4.1 ist eine direkte SSH-Verbindung zwischen `c1` und `s2` nicht zugelassen.

Hierfür wird für den Benutzer `jump` auf Server `s1` folgende Konfiguration hinterlegt bzw. der Datei `/etc/ssh/sshd_config` angefügt und gemäss Kapitel 4.2.4 übernommen:

```
# User jump ----------------------------------------------------------------
Match User jump
        DisableForwarding         no
        AllowTcpForwarding        yes
        PermitOpen                192.168.2.1:22
        MaxSessions               0
```

Quelltext 4.11: SSH-Serverdienst-Konfigurationszusatz für Jumphost auf Server `s1`

Mit dem `MaxSessions`-Parameter und Wert `0` wird bewirkt, dass sämtliche Shell-, Login- und Subsystem-Sessions deaktiviert werden, Forwarding jedoch immer noch erlaubt ist [49] [3].

Nun kann als Benutzer `user` auf dem Client-Host `c1` mittels `ssh`-Parameter `-J` oder `ProxyJump` (siehe Tabelle 3.1 in Kapitel 3.4.1.1) wie folgt ein Login mit dem Benutzer `cmd` auf dem Server `s2` über die Verbindung mit User `jump` auf Server `s1` ausgeführt werden:

```
c1$ ssh -J jump@192.168.1.1 cmd@192.168.2.1
Enter passphrase for key '/home/user/.ssh/id_ed25519':
jump@192.168.1.1's password:
Enter passphrase for key '/home/user/.ssh/id_ed25519':
cmd@192.168.2.1's password:
s2$
```

Quelltext 4.12: SSH-Login vom Client `c1` via Jumphost (Server `s1`) auf Server `s2`

Der vorherigen Ausgabe kann entnommen werden, dass für den Benutzer `jump` auf `s1` eine Public-Key- und Passwort-Authentisierung durchgeführt werden muss, worauf dasselbe nochmals für den Benutzer `cmd` auf `s2` zu tätigen ist. Mittels `PermitOpen`-Parameter in der Konfiguration von `s1` wird definiert, zu welchem Ziel ein Local Forwarding geöffnet werden darf.

[49] Channels vom Typ „TCP/IP Port Forwarding“ sind unabhängig von Sessions (siehe Kapitel 3.1.5.2)

4.2.7. Authentisierungs-Agent

In der Ausgabe des vorherigen Kapitels (Quelltext 4.12) sieht man, dass die Passphrase für den Private-Key bei jedem Login eingegeben werden muss. Um diese Eingabe zu reduzieren und trotzdem den Private-Key zu verwenden, kann dieser in den Authentisierungs-Agenten `ssh-agent` (siehe Kapitel 3.4.3.3) geladen werden.

Je nach Installation kann dieser bereits im Hintergrund gestartet sein, was z.B. bei einer Installation der Linux-Distribution „Debian“ mit graphischer Oberfläche der Fall ist. Hierfür ist die Umgebungsvariable `SSH_AUTH_SOCK` zu prüfen [50], welche bei einem hinterlegten Wert auf einen laufenden `ssh-agent` hinweist. Bei einer OpenBSD-Installation wie sie in Kapitel 4.1.2 vorgenommen wurde, ist der `ssh-agent` nicht aktiviert.

Die Manpage von `ssh-agent` [56] empfiehlt dessen Start mittels folgenden Befehls:
`eval $(ssh-agent -s)` [51]. Dies startet den `ssh-agent` für die aktive Shell-Sitzung, worauf dieser aufgrund der gewählten Start-Methode dann mit der Sitzung beendet wird.

Somit wird `ssh-agent` auf dem Client-Host `c1` wie folgt gestartet:

```
c1$ env | grep SSH_AUTH_SOCK
c1$ eval $(ssh-agent -s)
Agent pid 85923
c1$ env | grep SSH_AUTH_SOCK
SSH_AUTH_SOCK=/tmp/ssh-zLMNeLGPqpdh/agent.70143
```

Quelltext 4.13: Starten des Authentisierungs-Agenten `ssh-agent` in der aktiven Sitzung auf dem Client `c1`

Nun kann der zu verwendende Private-Key aus Kapitel 4.2.1 für die entsprechende Public-Key-Authentisierung mit dem Befehl `ssh-add` (siehe Kapitel 3.4.2.1) hinzugefügt werden:

```
c1\$ ssh-add /home/user/.ssh/id_ed25519
Enter passphrase for /home/user/.ssh/id_ed25519:
Identity added: /home/user/.ssh/id_ed25519 (user@c1.lab.internal)
```

Quelltext 4.14: Hinzufügen eines Private-Keys zum Authentisierungs-Agenten `ssh-agent` in der aktiven Sitzung auf dem Client `c1`

[50] z.B. mittels Ausführen von `env | grep SSH_AUTH_SOCK`
[51] Alternative Schreibweise: `eval `ssh-agent -s`` [76]

Mit geladenem Schlüssel im `ssh-agent` wird nach dessen Passphrase nicht mehr nachgefragt und der Schlüssel wird bei der nächsten Public-Key-Authentisierung automatisch verwendet.

Dies zeigt die erneute Anwendung der Jumphost-Verbindung gemäss Kapitel 4.2.6, Quelltext 4.12 mit Parameter `-v`:

```
c1$ ssh -v -J jump@192.168.1.1 cmd@192.168.2.1
...
debug1: Setting implicit ProxyCommand from ProxyJump: ssh -l jump -v -W '[%h
    ]:%p' 192.168.1.1
debug1: Executing proxy command: exec ssh -l jump -v -W '[192.168.2.1]:22'
    192.168.1.1
...
debug1: Authenticating to 192.168.1.1:22 as 'jump'
...
debug1: get_agent_identities: agent returned 1 keys
debug1: Will attempt key: /home/user/.ssh/id_ed25519 ED25519 SHA256:SQGlWRYdvL
    /NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk agent
...
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Offering public key: /home/user/.ssh/id_ed25519 ED25519 SHA256:
    SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk agent
debug1: Server accepts key: /home/user/.ssh/id_ed25519 ED25519 SHA256:
    SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk agent
Authenticated using "publickey" with partial success.
debug1: Authentications that can continue: password
debug1: Next authentication method: password
jump@192.168.1.1's password:
Authenticated to 192.168.1.1 ([192.168.1.1]:22) using "password".
...
debug1: Authenticating to 192.168.2.1:22 as 'cmd'
...
debug1: get_agent_identities: agent returned 1 keys
debug1: Will attempt key: /home/user/.ssh/id_ed25519 ED25519 SHA256:SQGlWRYdvL
    /NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk agent
...
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Offering public key: /home/user/.ssh/id_ed25519 ED25519 SHA256:
    SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk agent
debug1: Server accepts key: /home/user/.ssh/id_ed25519 ED25519 SHA256:
    SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk agent
Authenticated using "publickey" with partial success.
debug1: Authentications that can continue: password
debug1: Next authentication method: password
cmd@192.168.2.1's password:
Authenticated to 192.168.2.1 (via proxy) using "password".
...
s2$
```

Quelltext 4.15: SSH-Login via Jumphost (Server `s1`) auf Server `s2` mit geladenem Key im `ssh-agent` und Debugging-Ausgaben

```
c1$ ssh -J jump@192.168.1.1 cmd@192.168.2.1
jump@192.168.1.1's password:
cmd@192.168.2.1's password:
s2$
```

Quelltext 4.16: SSH-Login via Jumphost (Server `s1`) auf Server `s2` mit geladenem Key im `ssh-agent` ohne Debugging-Ausgaben

4.2.7.1. Agent-Forwarding

Mittels Agent-Forwarding kann die Verbindung zum Authentisierungs-Agenten via SSH weitergeleitet werden. Dies ermöglicht das Nutzen des Authentisierungs-Agenten vom Client `c1` in einer aktiven SSH-Sitzung auf dem Server `s1`, womit auf dem Server `s1` Schlüssel verwendet werden können, welche sich nur auf `c1` befinden.

Die Manpage des SSH-Clients `ssh` [47] erwähnt, dass Agent-Forwarding mit Vorsicht zu verwenden ist. Ein Angreifer, welcher die Dateiberechtigungen des Unix Sockets des `ssh-agent` auf dem entfernten Host umgehen kann, kann auf den weitergeleiteten Agenten zugreifen [47]. Er kann zwar keine Schlüssel auslesen, aber sie zur weiteren Authentisierung auf andere Systeme verwenden [47]. Als sicherere Alternative schlägt die SSH-Client-Manpage [47] die Verwendung eines Jumphosts (siehe Kapitel 4.2.6) vor.

In der Laborumgebung wird das Agent-Forwarding für den Benutzer `agent` auf dem Server `s1` durch Anfügen folgender Konfiguration zu `/etc/ssh/sshd_config` eingerichtet und die Konfiguration gemäss Kapitel 4.2.4 übernommen [52]:

```
# User agent ----------------------------------------------------------------
Match User agent
        AllowAgentForwarding      yes
        PermitTTY                 yes
        ForceCommand              none
```

Quelltext 4.17: SSH-Serverdienst-Konfigurationszusatz für Agent-Forwarding und Kommandozeilenzugriff auf Server `s1`

Nun kann die SSH-Verbindung zum Benutzer `agent` auf dem Server `s1` mit Agent-Forwarding aufgebaut werden [53]:

```
c1$ eval $(ssh-agent -s)
Agent pid 90654
c1$ ssh-add /home/user/.ssh/id_ed25519
Enter passphrase for /home/user/.ssh/id_ed25519:
Identity added: /home/user/.ssh/id_ed25519 (user@c1.lab.internal)
c1$ ssh -A agent@192.168.1.1
agent@192.168.1.1's password:
s1$ ssh-add -L
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIMH4AMuabHjbA5M8rXbOY2EEy0Da/
   ZOtM4psNvSO50pc user@c1.lab.internal
s1$ ssh cmd@192.168.2.1
cmd@192.168.2.1's password:
s2$
```

Quelltext 4.18: SSH-Login mit Agent-Forwarding auf Server `s1` mit geladenem Key im `ssh-agent` auf dem Client `c1` sowie SSH-Login mit dem Key aus dem `ssh-agent` auf `s2`

[52] Im Gegensatz zur Jumphost-Konfiguration ist hier die Option `DisableForwarding` mit Wert `no` nicht zu definieren, um ein funktionsfähiges Agent-Forwarding zu erhalten. Obwohl die Definition der Option auch Agent-Forwarding beinhaltet [3], ist der Grund dafür derzeit unbekannt

[53] Aktivierung / Deaktivierung des Agent-Forwarding im SSH-Client siehe Tabelle 3.1 in Kapitel 3.4.1.1

4.2.7.2. Einschränkung der Key-Nutzung

Damit die im Agent geladenen Schlüssel nur für die dafür bestimmten Ziele verwendet werden können, können beim Hinzufügen des Keys mit `ssh-add` (siehe Kapitel 3.4.2.1) Bedingungen hinzugefügt werden.

Mit folgendem Befehl wird ein Schlüssel zum `ssh-agent` hinzugefügt, sodass dieser nur für die Authentisierung...

- ... als Benutzer `agent` auf dem Server `s1` (`192.168.1.1`) ...
  `-h 'agent@192.168.1.1'`
- ... und bei Agent-Forwarding auf dem Server `s1` (`192.168.1.1`) als Benutzer `cmd` auf Server `s2` (`192.168.2.1`) ...
  `-h '192.168.1.1>cmd@192.168.2.1'`

... verwendet werden dürfen:
`ssh-add -h 'agent@192.168.1.1' -h '192.168.1.1>cmd@192.168.2.1' ~/.ssh/id_ed25519`

Die hinterlegten Hosts in den `-h`-Parametern müssen dem Client in dessen `known_hosts`-Datei mit entsprechenden Public-Keys unter `~/.ssh/known_hosts` [54] bekannt sein. Diese `known_hosts`-Datei wird z.B. beim ersten Login via SSH abgefüllt, womit der Client die Public-Keys des Zielhosts erhält.

In dieser Laborumgebung erhält der Client `c1` die Public-Keys von Server `s2` nur, wenn er mittels Jumphost darauf verbindet. Würde man von Server `s1` zu Server `s2` verbinden, würde die `known_hosts`-Datei auf `s1` bearbeitet werden.

Als Alternative können mit `ssh-keyscan` (siehe Kapitel 3.4.2.3) die Public-Keys ausgelesen und manuell in die entsprechende `known_hosts`-Datei gepflegt werden.

Um somit auf Client `c1` mit leeren `known_hosts`-Dateien den `ssh-add`-Befehl mit Host-Einschränkung durchführen zu können, kann folgender Ablauf durchgeführt werden:

```
c1$ ssh-keyscan 192.168.1.1 >> ~/.ssh/known_hosts
...
c1$ ssh cmd@192.168.1.1 ssh-keyscan 192.168.2.1 >> ~/.ssh/known_hosts
Enter passphrase for key '/home/user/.ssh/id_ed25519':
cmd@192.168.1.1's password:
...
c1$ cat ~/.ssh/known_hosts
192.168.1.1 ssh-rsa AAAA...OD8=
192.168.1.1 ecdsa-sha2-nistp256 AAAA...cUdw=
192.168.1.1 ssh-ed25519 AAAA...5P1D
192.168.2.1 ssh-rsa AAAA...9lk=
192.168.2.1 ecdsa-sha2-nistp256 AAAA...PUk=
192.168.2.1 ssh-ed25519 AAAA...0dy2
```

Quelltext 4.19: Auslesen der Public-Keys von Server `s1` und `s2` (via `s1`) in das `known_hosts`-Files des Benutzers `user` des Clients `c1`

Danach kann mit zuvor erwähntem Befehl der Private-Key so beim `ssh-agent` hinterlegt werden, dass er nur für die zuvor erwähnten Zwecke eingesetzt werden darf:
`ssh-add -h 'agent@192.168.1.1' -h '192.168.1.1>cmd@192.168.2.1' ~/.ssh/id_ed25519`

Nun gilt es zu beachten, dass die Verwendung des Keys für andere Zwecke vom Authentisierungs-Agenten abgelehnt wird, was zwar dem Zweck entspricht, für welchen man die Bedingung eingerichtet hat, es jedoch ein weiterer Stolperstein beim Troubleshooting einer fehlerhafter Konfiguration sein kann.

[54] oder weiteren Pfaden wie `/etc/ssh/ssh_known_hosts`, die in der `ssh-add`-Manpage [50] aufgeführt sind. Zusätzlich ist die Angabe eines anderen Files mit `-H hostkey_file` möglich

4.2.8. Implementation mit Zertifikaten und CA

Im ermittelten Minimalstandard in Kapitel 3.5.3 wird die Verwendung von Zertifikaten empfohlen, was die Sicherheit erhöht, aber auch eine gewisse Komplexität mit sich bringt. Dieses Kapitel zeigt den möglichst einfachen Aufbau mit einer dedizierten CA auf, um Zertifikate zur Authentisierung verwenden zu können.

4.2.8.1. Erstellung CA

Für die CA wird eine separate OpenBSD-VM installiert [55]. Darin wird mit folgendem Befehl ein Schlüsselpaar (Private- und Public-Key) des Typs Ed25519 (EdDSA mit „Safe Curve“ Curve25519) für die CA in einem eigenen Ordner generiert und die Berechtigungen für Gruppen und andere auf den Schlüssel entzogen:

```
ca$ mkdir ~/ca
ca$ cd ~/ca/
ca$ ssh-keygen -C CA -f ca -t ed25519
Generating public/private ed25519 key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ca
Your public key has been saved in ca.pub
The key fingerprint is:
SHA256:GwSQXlRbLXCg51XIVqJatsqKZL8JZxr91tBNRU5DS3Q CA
The key's randomart image is:
+--[ED25519 256]--+
|     .+o.+++BX E  |
|     . .o ==*o+   |
|    . .. B..oo    |
|     .  B o.      |
|       ..So       |
|    . ....o.      |
|  = + oo.         |
| o O +. .         |
|  o =o.           |
+----[SHA256]-----+
ca$ chmod 700 ca ca.pub
```

Quelltext 4.20: Erstellung CA mit Ed25519

Eine Passphrase ist entsprechend zu definieren, da jeder mit Zugang zum Private-Key der CA und entsprechender Passphrase Zertifikate generieren kann. Alternative Schlüsseltypen gemäss Minimalstandard (Kapitel 3.5.3) sind Kapitel 4.2.1 zu entnehmen.

Es wird empfohlen im Ordner der CA entsprechende Unterordner für Hosts und Benutzer zu erstellen, was in dieser Arbeit in zukünftigen Handhabungen mit der CA ersichtlich sein wird.

[55] In einer produktiven Umgebung ist die CA bestmöglich zu schützen, da beim Vertrauen einer CA sämtlichen davon abstammenden Zertifikaten vertraut wird. In einer Laborumgebung wie dieser könnte die CA auch auf einem bestehenden Server aufgebaut werden

4.2.8.2. Verteilung CA-Public-Key und Konfiguration

Der Public-Key der CA `ca.pub` ist nun auf den Servern `s1` und `s2` unter `/etc/ssh/ca.pub` zu hinterlegen [56] und mit folgenden Berechtigungen zu versehen:

```
sX$ doas chown root:wheel /etc/ssh/ca.pub
sX$ doas chmod 644 /etc/ssh/ca.pub
sX$ ls -l /etc/ssh/*.pub
-rw-r--r--  1 root  wheel   84 Aug 13 13:43 /etc/ssh/ca.pub
-rw-r--r--  1 root  wheel  182 Aug  9 10:14 /etc/ssh/ssh_host_ecdsa_key.pub
-rw-r--r--  1 root  wheel  102 Aug  9 10:14 /etc/ssh/ssh_host_ed25519_key.pub
-rw-r--r--  1 root  wheel  574 Aug  9 10:14 /etc/ssh/ssh_host_rsa_key.pub
```

Quelltext 4.21: Anpassen Berechtigungen des CA-Public-Keys auf den Servern `s1` und `s2`

Zusätzlich wird der hinterlegte CA-Public-Key in der Grundkonfiguration (siehe Kapitel 4.2.2) hinterlegt und die Zertifikatsauthentisierung entsprechend konfiguriert (siehe Abschnitt „Authentisierung und Autorisierung: Zertifikats-Authentisierung“ in Kapitel 3.4.3.1) und gemäss Kapitel 4.2.4 übernommen:

```
# Certificate Authentication ----------------------------------------------
TrustedUserCAKeys             /etc/ssh/ca.pub
AuthorizedPrincipalsFile      .ssh/authorized_principals
```

Quelltext 4.22: Zusatz zur SSH-Serverdienst-Grundkonfiguration `/etc/ssh/sshd_config` auf Server `s1` und `s2` zum Vertrauen des CA-Public-Keys

Ohne `AuthorizedPrincipalsFile` würde jedes Zertifikat unserer CA für den SSH-Login akzeptiert werden. Mit dieser Datei werden Principals hinterlegt, wobei einer davon in einem Zertifikat hinterlegt sein muss, damit das Zertifikat akzeptiert wird.

Dazu wird für jeden Benutzer `cmd`, `file`, `jump` und `agent` auf Server `s1` und Benutzer `cmd` auf Server `s2` die Datei `~/.ssh/authorized_principals` mit folgendem Inhalt erstellt:

```
principala
principalb
```

Quelltext 4.23: `AuthorizedPrincipalsFile` unter `~/.ssh/authorized_principals`

[56] Mittels Kopieren der Datei oder des Inhalts in eine neue Datei, es handelt sich hierbei um eine Textzeile

4.2.8.3. Signieren eines User-Keys

Der in Kapitel 4.2.1 generierte Public-Key auf dem Client `c1` des Benutzers `user` wird auf den CA-Server übertragen und mit folgendem Befehl mit einem Principal aus dem zuvor definierten `AuthorizedPrincipalsFile` und einem Identifikator versehen:

```
ca$ mkdir -p host/c1/user
ca$ cd host/c1/user/
ca$ ssh-keygen -s ~/ca/ca -n principalb -I user_c1_20230813_1231 id_ed25519.
   pub
Enter passphrase:
Signed user key id_ed25519-cert.pub: id "user_c1_20230813_1231" serial 0 for
   principalb valid forever
ca$ ls
id_ed25519-cert.pub id_ed25519.pub
```

Quelltext 4.24: Signieren eines Benutzer-Public-Keys mit entsprechendem Principal

Die einzugebende Passphrase entspricht dieser welche bei der CA-Erstellung hinterlegt wurde. Nun kann die neue Zertifikatsdatei (hier `id_ed25519-cert.pub` [57]) dem Client übergeben werden, welche dieser dann zur Authentisierung verwenden kann.

Der Zertifikatsinhalt kann mit `ssh-keygen -L -f id_ed25519-cert.pub` ausgegeben werden:

```
ca$ ssh-keygen -L -f id_ed25519-cert.pub
id_ed25519-cert.pub:
        Type: ssh-ed25519-cert-v01@openssh.com user certificate
        Public key: ED25519-CERT SHA256:SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/
           QJ+CEk
        Signing CA: ED25519 SHA256:GwSQXlRbLXCg51XIVqJatsqKZL8JZxr91tBNRU5DS3Q
            (using ssh-ed25519)
        Key ID: "user_c1_20230813_1231"
        Serial: 0
        Valid: forever
        Principals:
                principalb
        Critical Options: (none)
        Extensions:
                permit-X11-forwarding
                permit-agent-forwarding
                permit-port-forwarding
                permit-pty
                permit-user-rc
```

Quelltext 4.25: Ausgabe eines Zertifikatinhalts mit `ssh-keygen`

Wie der Ausgabe entnommen werden kann, ist es möglich bestimmte Funktionalitäten zur Verwendung von SSH zusammen mit dem Zertifikat zu erteilen oder entziehen. In diesem Falle wird ohne genauere Definition die unter `Extensions` aufgelisteten Funktionalitäten erteilt. Unter anderem kann dem Zertifikat eine Befehls-Forcierung (`force-command`) hinterlegt, Weiterleitungen aktiviert oder deaktiviert oder eine Adressliste mit gültigen Quell-Hosts (die das Zertifikat nutzen dürfen) hinterlegt werden. Ebenfalls kann den Zertifikaten auch eine bestimmte Gültigkeitsdauer hinterlegt werden, nach welcher diese nicht mehr gültig sind. Des Weiteren ist auch die Definition und Pflege von „Key Revocation Lists“ möglich, in welcher Schlüssel und Zertifikate für ungültig erklärt werden können.

Aus Zeitgründen wird auf ein tieferes Eintauchen in die Verwendung von Zertifikaten mit OpenSSH verzichtet und auf die Manpage von `ssh-keygen` [53] sowie das Buch *SSH Mastery - Second Edition* (2018) [60] verwiesen, welche dieses Gebiet genauer behandeln.

[57] Die CA kann trotz ihres Schlüssel-Typs von Ed25519 auch Public-Keys anderer Typen, z.B. RSA signieren

4.2.8.4. Verwenden eines Zertifikats

Nun stehen dem Benutzer `user` auf dem Client-Host `c1` folgende Dateien unter `~/.ssh` zur Verfügung:

- `id_ed25519`: In Kapitel 4.2.1 generierter Private-Key
- `id_ed25519.pub`: In Kapitel 4.2.1 generierter Public-Key
- `id_ed25519-cert.pub`: Zuvor generiertes Zertifikat durch Signieren des Public-Keys

Baut man mit dem SSH-Client `ssh` eine Verbindung auf, werden ohne Spezifizierung des Private-Keys mit `-i` folgende Pfade geprüft: `~/.ssh/id_rsa`, `~/.ssh/id_ecdsa`, `~/.ssh/id_ecdsa_sk`, `~/.ssh/id_ed25519`, `~/.ssh/id_ed25519_sk` und `~/.ssh/id_dsa` [47]. Zusätzlich wird geprüft, ob ein entsprechendes Zertifikat vorhanden ist, das dem Dateinamen des Private-Keys plus Suffix `-cert.pub` entspricht [58] und verwendet dieses. Andernfalls kann dem SSH-Client mit der Option `CertificateFile` ein bestimmtes Zertifikat mitgegeben werden.

Demnach werden mit den entsprechend hinterlegten Dateien nun beim SSH-Verbindungsaufbau auch die Zertifikate verwendet:

```
c1$ ssh -v cmd@192.168.1.1
OpenSSH_9.3, LibreSSL 3.7.2
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Connecting to 192.168.1.1 [192.168.1.1] port 22.
debug1: Connection established.
debug1: identity file /home/user/.ssh/id_ed25519 type 3
debug1: identity file /home/user/.ssh/id_ed25519-cert type 7
...
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Offering public key: /home/user/.ssh/id_ed25519 ED25519 SHA256:
    SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk explicit
debug1: Authentications that can continue: publickey
debug1: Offering public key: /home/user/.ssh/id_ed25519 ED25519-CERT SHA256:
    SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk explicit
debug1: Server accepts key: /home/user/.ssh/id_ed25519 ED25519-CERT SHA256:
    SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk explicit
Enter passphrase for key '/home/user/.ssh/id_ed25519':
Authenticated using "publickey" with partial success.
debug1: Authentications that can continue: password
debug1: Next authentication method: password
cmd@192.168.1.1's password:
Authenticated to 192.168.1.1 ([192.168.1.1]:22) using "password".
...
debug1: Remote: cert: key options: agent-forwarding port-forwarding pty user-
    rc x11-forwarding
debug1: Remote: principals: key options: agent-forwarding port-forwarding pty
    user-rc x11-forwarding
debug1: Remote: cert: key options: agent-forwarding port-forwarding pty user-
    rc x11-forwarding
debug1: Remote: principals: key options: agent-forwarding port-forwarding pty
    user-rc x11-forwarding
...
s1$
```

Quelltext 4.26: SSH-Login mit durch CA signiertem Zertifikat und Debugging-Ausgaben

[58] Welches durch die Generierung des Zertifikats mit `ssh-keygen` bereits entsprechend genannt wurde

Zudem wird nun betreffend Authentisierungs-Agent das Zertifikat für die entsprechende Public-Key-Authentisierung mit dem Befehl `ssh-add` (siehe Kapitel 3.4.2.1) ebenfalls hinzugefügt, sofern dies mit dem Suffix `-cert.pub` neben dem Private-Key aufzufinden ist:

```
c1$ ssh-add id_ed25519
Enter passphrase for id_ed25519:
Identity added: id_ed25519 (user@c1.lab.internal)
Certificate added: id_ed25519-cert.pub (user_c1_20230813_1231)
c1$ ssh-add -l
256 SHA256:SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk user@c1.lab.internal (
    ED25519)
256 SHA256:SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk user@c1.lab.internal (
    ED25519-CERT)
```

Quelltext 4.27: Hinzufügen eines Private-Keys inkl. Zertifikat zum Authentisierungs-Agenten `ssh-agent` in der aktiven Sitzung auf dem Client `c1`

4.2.8.5. Forcieren der Zertifikatsauthentisierung

Mit der bisherigen Konfiguration kann die Public-Key-Authentisierung ohne Zertifikate dennoch durchgeführt werden, weshalb in der SSH-Server-Konfiguration die Zeile mit `AuthorizedKeysFile` wie folgt anzupassen und zu übernehmen ist:

```
AuthorizedKeysFile              none
```

Quelltext 4.28: Anpassung der SSH-Serverdienst-Grundkonfiguration `/etc/ssh/sshd_config` auf Server `s1` und `s2` zum Forcieren der Zertifikatsauthentisierung

Die `AuthorizedKeysFile` sämtlicher Benutzer auf den Servern `s1` und `s2` können nun gelöscht werden: `doas rm /home/*/.ssh/authorized_keys`

4.2.8.6. Host-Zertifikate

Die Host-Zertifikate der Server `s1` und `s2` können ebenfalls mit einem Zertifikat versehen werden. Dazu müssen die entsprechenden Public-Keys zur CA kopiert (nach `~/ca/host/s1` bzw. `~/ca/host/s2`) und mittels folgendem Befehl signiert werden:

```
ca$ cd ~/ca/host/sX/
ca$ ls
ssh_host_ecdsa_key.pub      ssh_host_ed25519_key.pub    ssh_host_rsa_key.pub
ca$ ssh-keygen -s ~/ca/ca -I host_sX -h ssh_host_*.pub
Enter passphrase:
Signed host key ssh_host_ecdsa_key-cert.pub: id "host_sX" serial 0 valid
   forever
Signed host key ssh_host_ed25519_key-cert.pub: id "host_sX" serial 0 valid
   forever
Signed host key ssh_host_rsa_key-cert.pub: id "host_sX" serial 0 valid forever
ca$ ls
ssh_host_ecdsa_key-cert.pub        ssh_host_ed25519_key-cert.pub
   ssh_host_rsa_key-cert.pub
ssh_host_ecdsa_key.pub             ssh_host_ed25519_key.pub
   ssh_host_rsa_key.pub
```

Quelltext 4.29: Signieren von Host-Public-Keys

Nun sind die `*-cert.pub`-Dateien auf beide Server nach `/etc/ssh/` zu kopieren und dort die Berechtigungen entsprechend anzupassen:

```
sX$ doas chown root:wheel /etc/ssh/ssh_host*-cert.pub
sX$ doas chmod 644 /etc/ssh/ssh_host*-cert.pub
sX# ls -l /etc/ssh/*.pub
-rw-r--r--  1 root  wheel   84 Aug 13 13:43 /etc/ssh/ca.pub
-rw-r--r--  1 root  wheel  543 Aug 13 15:21 /etc/ssh/ssh_host_ecdsa_key-cert.
   pub
-rw-r--r--  1 root  wheel  182 Aug  9 10:14 /etc/ssh/ssh_host_ecdsa_key.pub
-rw-r--r--  1 root  wheel  463 Aug 13 15:21 /etc/ssh/ssh_host_ed25519_key-cert
   .pub
-rw-r--r--  1 root  wheel  102 Aug  9 10:14 /etc/ssh/ssh_host_ed25519_key.pub
-rw-r--r--  1 root  wheel  935 Aug 13 15:21 /etc/ssh/ssh_host_rsa_key-cert.pub
-rw-r--r--  1 root  wheel  574 Aug  9 10:14 /etc/ssh/ssh_host_rsa_key.pub
```

Quelltext 4.30: Anpassen Berechtigungen des Zertifikate auf den Servern `s1` und `s2`

Die Zertifikate sind der SSH-Server-Grundkonfiguration im Bereich mit den `HostKey`-Parametern hinzuzufügen und gemäss Kapitel 4.2.4 zu übernehmen:

```
# Private keys of server ----------------------------------------------------
# Generated RSA server host key is 3072 bit
HostKey                         /etc/ssh/ssh_host_rsa_key
HostCertificate                 /etc/ssh/ssh_host_rsa_key-cert.pub
HostKey                         /etc/ssh/ssh_host_ecdsa_key
HostCertificate                 /etc/ssh/ssh_host_ecdsa_key-cert.pub
HostKey                         /etc/ssh/ssh_host_ed25519_key
HostCertificate                 /etc/ssh/ssh_host_ed25519_key-cert.pub
```

Quelltext 4.31: Erweiterung der SSH-Serverdienst-Grundkonfiguration `/etc/ssh/sshd_config` auf Server `s1` und `s2` mit entsprechenden Host-Zertifikaten

Auf dem Client-Host ist der Inhalt des CA-Public-Keys unter `~/ca/ca.pub` im `known_hosts`-File [59] wie folgt zu hinterlegen [54]:

```
@cert-authority * ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIOFmYpWgT2p87PlGZxqHq+
    X6EkJdYh3wo5exIed7OGfz CA
```

Quelltext 4.32: Hinterlegung des CA-Public-Keys im `known_hosts`-File des Clients

Anstelle von `*` kann eine Komma-separierte Liste an Mustern zum abgleichen entsprechender Hostnamen hinterlegt werden, um die Gültigkeit des CA-Public-Keys für bestimmte Hosts einzuschränken. Nun ist keine Liste an bekannten Host-Public-Keys mehr zu pflegen, ausser es ist die Verwendung eines Authentisierungsagenten mit Keys mit eingeschränkter Nutzung gemäss Kapitel 4.2.7.2 vorgesehen [60].

[59] Benutzerspezifisch unter `~/.ssh/known_hosts` oder systemweit unter `/etc/ssh/ssh_known_hosts` [47]

[60] Eventuell ist dies dennoch möglich bestimmte Ziele als Bedingung für Keys für die Verwendung mit `ssh-agent` zu hinterlegen, ohne dass diese explizit im `known_hosts`-File mitgegeben werden müssen, jedoch konnte das aus Zeitgründen im Rahmen dieser Arbeit nicht ermittelt werden

4.2.9. SSHFP-DNS-Records

Wie unter anderem in Kapitel 3.2 erwähnt wurde, kann der Fingerprint eines Public-Keys mittels DNS-Resource-Record des Typs „SSHFP“ zur Verifikation publiziert werden.

4.2.9.1. Aufbau DNS-Server und Einbau SSHFP-DNS-Records

In der Laborumgebung dieser Arbeit wurde kein DNS-Server in Betrieb genommen [61], weshalb auf Server `s1` ein einfacher DNS-Server mit dem in OpenBSD enthaltenen „Name Server Daemon“ `nsd` wie folgt aufgebaut wird [151]:

Die Zonen-Konfiguration der Domäne „lab.internal“ sowie die Reverse-Zone für „192.168.0.0/16“ wird der `nsd`-Konfiguration unter `/var/nsd/etc/nsd.conf` wie folgt angefügt:

```
zone:
        name: lab.internal
        zonefile: master/lab.internal.zone

zone:
        name: 168.192.in-addr.arpa
        zonefile: master/168.192.in-addr.arpa.zone
```

Quelltext 4.33: Erweiterung der `nsd`-Konfiguration unter `/var/nsd/etc/nsd.conf` mit der Zone „lab.internal“ auf Server `s1`

Die DNS-„SSHFP“-Einträge können den Servern `s1` und `s2` jeweils anhand dessen Host-Public-Keys wie folgt ausgelesen werden:

```
s1$ ssh-keygen -r s1 -f /etc/ssh/ssh_host_rsa_key.pub
s1 IN SSHFP 1 1 df51d567a511ed952f2e0e6142d333b69de15f54
s1 IN SSHFP 1 2 87
   f46f6f5121ed6c7de7c3257920437c9ab39cbea437406b869f38884ab0fbcf
s1$ ssh-keygen -r s1 -f /etc/ssh/ssh_host_ecdsa_key.pub
s1 IN SSHFP 3 1 a4066d94112c1fc1d1b89d3d8c9797f246c77842
s1 IN SSHFP 3 2 6
   f692ae2a46e51194d547b57a9acc60c05900a1014340e933838fd7f6bf75435
s1$ ssh-keygen -r s1 -f /etc/ssh/ssh_host_ed25519_key.pub
s1 IN SSHFP 4 1 25d2a14452b3fec5dfaf1b599670c807947e2bb7
s1 IN SSHFP 4 2 72
   d5a9721efc13c57ca9912fef8bfbd8741a1d54dab82801d31179c45d4ebd15

s2$ ssh-keygen -r s2 -f /etc/ssh/ssh_host_rsa_key.pub
s2 IN SSHFP 1 1 75d131a02ee03784a387e332fa567baea27e48ea
s2 IN SSHFP 1 2
   ec9b3e714afe381284b968cd2c2df48d332481ac6e88cbfb9468453c7d3ba4de
s2$ ssh-keygen -r s2 -f /etc/ssh/ssh_host_ecdsa_key.pub
s2 IN SSHFP 3 1 76141e6c4c399738ebce01d5397737ebc1c32e1c
s2 IN SSHFP 3 2 0
   c7975e396a42c2fa3708d3118de826f1c4aab0034d88be0239894317d6ca34b
s2$ ssh-keygen -r s2 -f /etc/ssh/ssh_host_ed25519_key.pub
s2 IN SSHFP 4 1 4e90a37f94c1bd1c8b8fd09b2bcf713ab34fd740
s2 IN SSHFP 4 2
   cc4233ba4d31f004f59f2f078670f76c84af379c1757233d12ff1e7f68686e11
```

Quelltext 4.34: Auslesen der DNS-„SSHFP“-Einträge auf den Servern `s1` und `s2`

Bei den Werten nach „SSHFP“ steht der erste Wert für den Algorithmus (`1` für RSA, `3` für ECDSA und `4` für Ed25519) und der zweite Wert für den Hash-Algorithmus (`1` für SHA-1 und `2` für SHA-256) [18][38][39]. Demnach sind jeweils die Zeilen mit SHA-256 für die DNS-Zonendatei zu nehmen.

[61] Der DNS-Resolver der Firewall wird für den Internetzugang in Kapitel 4.1.2 eingebunden, jedoch ermöglicht dieser kein Hinterlegen von SSHFP-DNS-Records

Die DNS-Zone für „lab.internal“ wird mit folgendem Inhalt unter `/var/nsd/zones/master/lab.internal.zone` auf Server `s1` angelegt [152]:

```
$ORIGIN lab.internal.
$TTL 86400

lab.internal. IN SOA s1.lab.internal. admin.lab.internal. (
              2023081301 ; Serial
              7200       ; Refresh
              600        ; Expiry
              3600000    ; Expire
              60         ; Minimum
)

              IN NS           s1.lab.internal.

s1            IN A            192.168.1.1
s1            IN SSHFP    1 2 87
   f46f6f5121ed6c7de7c3257920437c9ab39cbea437406b869f38884ab0fbcf
s1            IN SSHFP    3 2 6
   f692ae2a46e51194d547b57a9acc60c05900a1014340e933838fd7f6bf75435
s1            IN SSHFP    4 2 72
   d5a9721efc13c57ca9912fef8bfbd8741a1d54dab82801d31179c45d4ebd15

s2            IN A            192.168.2.1
s2            IN SSHFP    1 2
   ec9b3e714afe381284b968cd2c2df48d332481ac6e88cbfb9468453c7d3ba4de
s2            IN SSHFP    3 2 0
   c7975e396a42c2fa3708d3118de826f1c4aab0034d88be0239894317d6ca34b
s2            IN SSHFP    4 2
   cc4233ba4d31f004f59f2f078670f76c84af379c1757233d12ff1e7f68686e11

c1            IN A            192.168.100.1
```

Quelltext 4.35: DNS-Zonendatei für „lab.internal“ unter `/var/nsd/zones/master/lab.internal.zone` auf dem Server `s1`

Die DNS-Reverse-Zone für „192.168.0.0/16“ bzw. „168.192.in-addr.arpa“ wird mit folgendem Inhalt unter `/var/nsd/zones/master/168.192.in-addr.arpa.zone` auf Server `s1` angelegt [152]:

```
$ORIGIN 168.192.in-addr.arpa.
$TTL 86400

168.192.in-addr.arpa. IN SOA s1.lab.internal. admin.lab.internal (
              2023081301 ; Serial
              7200       ; Refresh
              600        ; Expiry
              3600000    ; Expire
              60         ; Minimum
)

              IN NS           s1.lab.internal.

1.1           IN PTR          s1.lab.internal.

1.2           IN PTR          s2.lab.internal.

1.100         IN PTR          c1.lab.internal.
```

Quelltext 4.36: DNS-Zonendatei für „168.192.in-addr.arpa“ unter `/var/nsd/zones/master/168.192.in-addr.arpa.zone` auf dem Server `s1`

Danach wird der `nsd`-Daemon mittels `doas rcctl enable nsd` aktiviert und mit `doas rcctl start nsd` auf Server `s1` gestartet und die Zonen mit `doas nsd-control reload lab.internal` und `doas nsd-control reload 168.192.in-addr.arpa` neu geladen.

Nun ist der DNS-Server `s1` entsprechend in den Systemen zu hinterlegen [62].

4.2.9.2. Verifikation mittels SSH-Client

Mit der Konfiguration des DNS-Servers ergibt sich der Vorteil, dass nun SSH-Verbindungen nicht mehr über die IP-Adresse sondern über dessen Domain-Namen aufgerufen werden können.

Standardmässig wird vom SSH-Client der SSHFP-DNS-Record nicht geprüft, da dessen Option `VerifyHostKeyDNS` mit der Einstellung `no` ausgeliefert wird [57]. Über die Kommandozeile kann mittels Parameter `-o "VerifyHostKeyDNS yes"` die Verifikation mittels SSHFP-DNS-Record aktiviert werden, wobei diese bei einer Angabe des Ziels mittels IP-Adresse übersprungen wird.

```
c1$ ssh -v -o "VerifyHostKeyDNS yes" cmd@192.168.1.1
...
debug1: Authenticating to 192.168.1.1:22 as 'cmd'
...
debug1: Server host certificate: ssh-ed25519-cert-v01@openssh.com SHA256:
   ctWpch78E8V8qZEv74v72HQaHVTauCgB0xF5xF1OvRU, serial 0 ID "host_s1" CA ssh-
   ed25519 SHA256:GwSQXlRbLXCg51XIVqJatsqKZL8JZxr91tBNRU5DS3Q valid forever
debug1: skipped DNS lookup for numerical hostname
...
```

Quelltext 4.37: Ausführung des SSH-Client mit aktivierter SSHFP-DNS-Record-Abfrage mit einer IP-Adresse als Ziel und Debugging-Meldungen

Wird der Domain-Name dies Zielhosts angegeben, kann die Verifikation eingesehen werden:

```
c1$ ssh -v -o "VerifyHostKeyDNS yes" cmd@s1.lab.internal
...
debug1: Authenticating to s1.lab.internal:22 as 'cmd'
...
debug1: Server host certificate: ssh-ed25519-cert-v01@openssh.com SHA256:
   ctWpch78E8V8qZEv74v72HQaHVTauCgB0xF5xF1OvRU, serial 0 ID "host_s1" CA ssh-
   ed25519 SHA256:GwSQXlRbLXCg51XIVqJatsqKZL8JZxr91tBNRU5DS3Q valid forever
debug1: found 3 insecure fingerprints in DNS
debug1: verify_host_key_dns: matched SSHFP type 4 fptype 2
debug1: matching host key fingerprint found in DNS
...
```

Quelltext 4.38: Ausführung des SSH-Client mit aktivierter SSHFP-DNS-Record-Abfrage mit Domain-Namen als Ziel und Debugging-Meldungen

Hier wird vom SSH-Client darauf hingewiesen, dass es sich um „unsichere“ Einträge handelt, was bedeutet, dass sie nicht mit DNSSEC abgesichert sind [63].

Es ist zu beachten, dass beim Prüfen SSHFP-DNS-Records vom SSH-Client die Verbindung nur nicht aufgebaut wird, denn der hinterlegte Fingerprint nicht übereinstimmt. Ist hingegen für den entsprechenden Zielhost kein SSHFP-DNS-Record hinterlegt oder die Zieladresse eine IP-Adresse, fährt der SSH-Client trotz gesetzter Option (`-o "VerifyHostKeyDNS yes"`) fort.

[62] Üblicherweise wird dies in der Datei `/etc/resolv.conf` getätigt, jedoch wird in dieser Laborumgebung die Firewall als DNS-Resolver verwendet (siehe Kapitel 4.1.2), auf welchem nun mittels „Domain Override“-Einstellung für die Domänen „lab.internal“ und „168.192.in-addr.arpa“ auf den Server `s1` unter `192.168.1.1` verwiesen wird

[63] Diese Absicherung liegt ausserhalb dem Rahmen dieser Arbeit

4.2.10. Client-Verifikation mittels DNS

Durch Anpassung der Option `UseDNS` von `no` zu `yes` in der SSH-Server-Grundkonfiguration (siehe Kapitel 4.2.2) und Übernahme gemäss Kapitel 4.2.4 wird auf dem SSH-Server die Verwendung von DNS aktiviert.

Dies ermöglicht die Verwendung des Domain-Names für `Match`-Einträge in der SSH-Server-Konfiguration, was folgende Ausgabe vorzeigt.

```
...
debug3: Trying to reverse map address 192.168.100.1.
debug2: parse_server_config_depth: config reprocess config len 5632
debug3: checking match for 'User cmd' user cmd host c1.lab.internal addr
    192.168.100.1 laddr 192.168.1.1 lport 22
debug1: user cmd matched 'User cmd' at line 112
debug3: match found
debug3: reprocess config:113 setting PermitTTY yes
debug3: reprocess config:114 setting ForceCommand none
...
```

Quelltext 4.39: Ausgabe von Debugging-Meldungen von `sshd` während eines Logins

Zusätzlich überprüft der SSH-Server beim Verbindungsaufbau mittels DNS den Domain-Namen des Clients (Reverse-DNS-Abfrage mit der Client-IP-Adresse) und verifiziert diesen mittels DNS-Abfrage (nach der IP-Adresse mit dem erhaltenen Domain-Namens), ob die erhaltene IP-Adresse zum Hostnamen der des Clients entspricht [3]. Stimmen Client-IP-Adresse und dessen Eintrag via DNS nicht überein wird die Verbindung trotzdem zugelassen, aber ein entsprechender Eintrag unter `/var/log/authlog` geloggt:

```
sshd[90015]: Address 192.168.100.1 maps to c1.lab.internal, but this does not
    map back to the address.
sshd[90015]: Accepted password for cmd from 192.168.100.1 port 5578 ssh2
```

Quelltext 4.40: Log-Eintrag von `sshd` in `/var/log/authlog` bei einer fehlerhaften Verifizierung des Clients mittels DNS

4.2.11. FIDO2 Authentisierung mit YubiKey

Innerhalb dieses Kapitels wird zur Authentisierung mit FIDO2 ein YubiKey verwendet. Der YubiKey ist, sofern nicht anders erwähnt, per USB am Client-Host `c1` angebunden.

Für die FIDO2-SSH-Authentisierung existieren zwei Typen von Anmeldedaten / „Credentials“ [153]:

- „Non-Discoverable Credentials“, welche zusätzlich zum YubiKey die FIDO2-Credential-ID im Ordner `~/.ssh` des Clients in einer Datei hinterlegt haben muss
- „Discoverable (resident) Credentials“, welche nicht auf die Hinterlegung der Credential-ID auf dem Client angewiesen ist

Obwohl ein „Non-Discoverable Credential“ hier einen zusätzlichen Faktor zur Absicherung bietet, macht dies den YubiKey weniger portabel [153]. Zudem kann die Credential-ID auf öffentlichen Computern einfacher eingesehen werden, weshalb die Verwendung von „Discoverable (resident) Credentials“ empfohlen und in diesem Kapitel behandelt wird [153]. Für beide Typen muss für ein erfolgreicher Login der YubiKey, ein zugehöriger PIN-Code und die Private-Key-Referenzdatei (wird später genauer ausgeführt) vorhanden sein.

Als Vorbedingungen für die Einrichtung muss folgendes erfüllt sein [153]:

- FIDO2-PIN auf dem YubiKey muss mittels YubiKey Manager [154] definiert sein [64]
- In der SSH-Server-Konfiguration ist die Option `PubkeyAuthOptions verify-required` hinterlegt und gemäss Kapitel 4.2.4 übernommen, damit OpenSSH bei der Authentisierung zusätzlich den FIDO2-PIN abfragt und nicht nur nach einer Berührung des YubiKeys (mit `PubkeyAuthOptions touch-required`) fragt (siehe Abschnitt „Authentisierung und Autorisierung: Public-Key-Authentisierung“ in Kapitel 3.4.3.1) [65]

[64] Dies wird in der Anwendung „YubiKey Manager“ [154] mit angeschlossenem YubiKey unter „Applications“ → „FIDO2“ → „Set PIN“ durchgeführt. Es wird empfohlen dies an einem System mit einer graphischen Oberfläche durchzuführen, wobei nach Definition des FIDO2-PINs der YubiKey wieder am Client `c1` angeschlossen wird

[65] Diese Einstellung ist in der SSH-Server-Grundkonfiguration (siehe Kapitel 4.2.2) auf Server `s1` und `s2` bereits hinterlegt

4.2.11.1. Schlüssel erstellen und auslesen

Auf dem Client-Host `c1` wird mit angeschlossenem YubiKey ähnlich wie in Kapitel 4.2.1 ein Schlüssel des Typs Ed25519 `ed25519-sk` [66] (EdDSA mit „Safe Curve“ Curve25519) erstellt [53][153]:

```
c1$ ssh-keygen -t ed25519-sk -O resident -O application=ssh:IDENTIFIKATOR -O
   verify-required
Generating public/private ed25519-sk key pair.
You may need to touch your authenticator to authorize key generation.
Enter PIN for authenticator:
You may need to touch your authenticator again to authorize key generation.
Enter file in which to save the key (/home/user/.ssh/id_ed25519_sk):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_ed25519_sk
Your public key has been saved in /home/user/.ssh/id_ed25519_sk.pub
The key fingerprint is:
SHA256:jINVYXxXhawwY5xILFTzX0YKIgS0xQNpTvn7r9H8IRQ user@c1.lab.internal
The key's randomart image is:
+[ED25519-SK 256]-+
|    .+O==O+.. ooo.|
|     =o++++O..oo  |
|    +...o oE=..o  |
|     .o.o   o.o   |
|     . o.S . .    |
|       .. +       |
|        .. + .    |
|         .. o .   |
|         .o. .    |
+----[SHA256]-----+
c1$
```

Quelltext 4.41: Erstellung eines Ed25519-Schlüssels mit `ssh-keygen` auf einem angeschlossenen FIDO Authenticator bzw. YubiKey

Mit dem optionalen Zusatz-Parameter `-O application=ssh:IDENTIFIKATOR` kann eine Identifikation für den Schlüssel hinterlegt werden, welche mit `ssh:` beginnen muss [53]. Mit unterschiedlichen Identifikatoren können mehrere Schlüssel auf demselben YubiKey hinterlegt werden. Mittels der Option `-O resident` wird mitgeteilt, dass der Schlüssel auf dem FIDO Authenticator (hier der YubiKey) gespeichert wird [53]. Der Parameter `-O verify-required` besagt, dass der Schlüssel mit einem FIDO2-PIN bei der Verwendung bestätigt werden muss [67] [53].

Alternativ ist zur Erfüllung des Minimalstandards aus Kapitel 3.5.3 die Wahl des Schlüssel-Typs `ecdsa-sk` (ECDSA) ebenfalls eine Option.

Wie der vorherigen Ausgabe zu entnehmen ist, muss während der Schlüsselerstellung der YubiKey berührt sowie der hinterlegte FIDO2-PIN eingegeben werden.

Zusätzlich wird der Public-Key unter `/home/user/.ssh/id_ed25519_sk.pub` zur Hinterlegung auf dem Zielhost gemäss Kapitel 4.2.1 abgespeichert und die erwähnte Private-Key-Referenzdatei unter `/home/user/.ssh/id_ed25519_sk` abgelegt. Beide Dateien müssen zwar zur SSH-Authentisierung vorhanden sein, können aber auch vom Host gelöscht und jederzeit vom YubiKey ausgelesen werden. Die Passphrase, die während der Schlüsselerstellung angegeben werden kann, ist für die Verwendung der Private-Key-Referenzdatei. Da die Private-Key-Referenzdatei ohne YubiKey und FIDO2-PIN keinen Nutzen hat, könnte diesem Vorgang die Passphrase ohne eine erhebliche Erhöhung des Sicherheitsrisikos ausgelassen werden.

[66] Die Schlüssel-Typen `ed25519-sk` und `ecdsa-sk` sind für die Verwendung von FIDO Authenticators vorgesehen [53][63]

[67] Ansonsten wird der Schlüssel bei einem SSH-Server mit der Einstellung `PubkeyAuthOptions verify-required` nicht akzeptiert, da bei der Verwendung nach keinem PIN gefragt wird

Die Private-Key-Referenzdateien und Public-Keys können mittels `ssh-keygen -K` von einem YubiKey ausgelesen werden, welche nach dessen Berührung und Eingabe des FIDO2-PINs in das derzeitige Arbeitsverzeichnis entsprechend ihrem Identifikator (sofern mittels `-O application=ssh:IDENTIFIKATOR` mitgegeben) gespeichert werden. Ebenso wird dort nochmals die Möglichkeit geboten, eine Passphrase für die Private-Key-Referenzdatei(en) zu hinterlegen [68]

```
c1$ ssh-keygen -K
Enter PIN for authenticator:
You may need to touch your authenticator to authorize key download.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Saved ED25519-SK key to id_ed25519_sk_rk
Saved ED25519-SK key ssh:test to id_ed25519_sk_rk_test
c1$ ls -l
total 16
-rw-------  1 user  wheel  496 Aug 14 23:13 id_ed25519_sk_rk
-rw-r--r--  1 user  wheel  133 Aug 14 23:13 id_ed25519_sk_rk.pub
-rw-------  1 user  wheel  501 Aug 14 23:13 id_ed25519_sk_rk_test
-rw-r--r--  1 user  wheel  141 Aug 14 23:13 id_ed25519_sk_rk_test.pub
```

Quelltext 4.42: Auslesen von je 2 Public-Keys und Private-Key-Referenzdateien (mit Identifikator „ssh:test“ und ohne Identifikator) aus dem YubiKey mittels `ssh-keygen -K`

Nun kann, sofern der entsprechende Public-Key auf bei dem Ziel-Benutzer und -Host hinterlegt ist [69], der passende Schlüssel für den SSH-Login mittels Parameter `-i` verwendet werden:

```
c1$ ssh -i id_ed25519_sk_rk cmd@192.168.1.1
Confirm user presence for key ED25519-SK SHA256:8yVsen/W+
   VxlZfoA0vneLrCZnYjGwMGEbDEKywH5dk4
Enter PIN for ED25519-SK key id_ed25519_sk_rk:
Confirm user presence for key ED25519-SK SHA256:8yVsen/W+
   VxlZfoA0vneLrCZnYjGwMGEbDEKywH5dk4
User presence confirmed
cmd@192.168.1.1's password:
s1$
```

Quelltext 4.43: SSH-Login mittels YubiKey

Nun wird gemäss Grundkonfiguration in Kapitel 4.2.2 zusätzlich zur Public-Key-Authentisierung, welche nun mit mehreren Faktoren durchgeführt wird, die Passwort-Authentisierung [70] durchgeführt. Ändert man in der SSH-Server-Konfiguration unter `/etc/ssh/sshd_config` die Zeile `AuthenticationMethods publickey,password` zu `AuthenticationMethods publickey` und übernimmt die Konfiguration gemäss Kapitel 4.2.4, wird nur noch die Public-Key-Authentisierung durchgeführt.

[68] Hierbei wird bei mehreren Schlüsseln für jede Private-Key-Referenzdatei dieselbe Passphrase hinterlegt

[69] Zum Beispiel durch Definition von `AuthorizedKeysFile .ssh/authorized_keys` in der entsprechenden SSH-Server-Konfiguration und Ablage des Public-Keys in der zugehörigen Datei `~/.ssh/authorized_keys`, siehe Kapitel 4.2.1

[70] Entspricht der Abfrage gemäss `cmd@192.168.1.1's password:` aus Quelltext 4.43

4.2.11.2. Anwendung von Zertifikaten und CA

In Kapitel 4.2.8 wird eine CA aufgebaut und Zertifikate ausgestellt, welche nun auch mit einem YubiKey verwendet werden sollen. Dies wird entsprechend Kapitel 4.2.8.3 durchgeführt mit dem Unterschied, dass der entsprechende Public-Key zuvor vom YubiKey mittels `ssh-keygen -K` auslesen wird. Hierfür wird der YubiKey am CA-Server angeschlossen.

```
ca$ mkdir /home/user/ca/yubikey
ca$ cd ca/yubikey/
ca$ ssh-keygen -K
Enter PIN for authenticator:
You may need to touch your authenticator to authorize key download.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Saved ED25519-SK key to id_ed25519_sk_rk
ca$ ssh-keygen -s ~/ca/ca -n principala -I yubikey_20230814_2159
   id_ed25519_sk_rk.pub
Enter passphrase:
Signed user key id_ed25519_sk_rk-cert.pub: id "yubikey_20230814_2159" serial 0
    for principala valid forever
```

Quelltext 4.44: Signieren eines Benutzer-Public-Keys eines YubiKeys

Damit wird das Zertifikat als Datei `id_ed25519_sk_rk-cert.pub` abgelegt. Diese Datei muss beim SSH-Client neben der vom YubiKey ausgelesenen Private-Key-Referenzdatei und Public-Keys (mittels `ssh-keygen -K`) liegen und dem Namen dieser Referenzdatei mit dem Suffix `-cert.pub` entsprechen. Somit ist das Zertifikat in diesem Falle separat zum YubiKey mitzuführen.

Im Rahmen dieser Arbeit wurde aus Zeitgründen das Abspeichern des Zertifikats in der gegebenen Form auf dem YubiKey nach einigen Versuchen nicht weiter verfolgt. Eventuell ist hierfür der YubiKey nicht als FIDO Authenticator sondern als Smartcard gemäss Artikel mit dem Titel „*Using PIV for SSH through PKCS #11*“ [155] einzurichten. Dass das Zertifikat nicht auf dem YubiKey mitgeführt wird, kann als Vorteil gesehen werden, bei welchem ein Angreifer im Besitz von YubiKey und zugehörigem FIDO2-PIN ohne Zertifikat keine erfolgreiche Authentisierung durchführen kann [71].

[71] Sofern die SSH-Server-Konfiguration dieser aus Kapitel 4.2.8.5 entspricht und somit keine `AuthorizedKeysFile`-Dateien beachtet

4.2.11.3. Versuch mit dem Authentisierungs-Agenten

Neben dem erfolglosen Versuch, das Zertifikat zu einem Private-Key eines FIDO Authenticators (hier YubiKey) ebenfalls auf diesem YubiKey zu hinterlegen, konnten im Rahmen dieser Arbeit ebenfalls keine Erfolge mit der Kombination eines FIDO Authenticators und dem Authentisierungs-Agenten `ssh-agent` erzielt werden.

Mit den gegebenen Mitteln und Versuchen mit diversen Konfigurationen [72] konnte in der gegebenen Zeit keine erfolgreiche Authentisierung erzielt werden, bei dem zuvor die Schlüssel eines FIDO Authenticators in den `ssh-agent` geladen wurden. Sobald ein solcher Schlüssel in den Authentisierungs-Agenten geladen und ein SSH-Login versucht wurde, erscheint folgende Fehlermeldung: [73]

```
1 c1$ ssh cmd@192.168.1.1
2 sign_and_send_pubkey: signing failed for ED25519-SK "" from agent: agent
     refused operation
3 cmd@192.168.1.1: Permission denied (publickey).
```

Quelltext 4.45: Fehlermeldung bei SSH-Login mit in den Authentisierungs-Agenten geladenen Schlüssel von einem YubiKey

Andere Benutzer im Internet scheinen dasselbe Problem, aber auch die entsprechende Authentisierung auf einem anderen Betriebssystem als OpenBSD mit OpenSSH in Gang gebracht zu haben [156]. Aus Zeitgründen wird diese Thematik im Rahmen dieser Arbeit nicht weiter verfolgt.

[72] Unter anderem der SSH-Server-Option `PubkeyAuthOptions` (siehe Abschnitt „Authentisierung und Autorisierung: Public-Key-Authentisierung“ in Kapitel 3.4.3.1) und diversen mit `ssh-keygen` generierten Schlüsseln und unterschiedlichen Parametern

[73] Wird die von YubiKey ausgelesene Private-Key-Referenzdatei wie z.B. `id_ed25519_sk_rk` mit `ssh`-Parameter `-i` mitgegeben, erscheint dieselbe Meldung mit angegebenem Pfad in der Fehlermeldung innerhalb der Anführungszeichen (`""`). Die SSH-Server-Einstellung `AuthorizedKeysFile` sowie die entsprechende Datei selber wurden mehrmals geprüft, wobei der entsprechende Public-Key hinterlegt ist

4.3. Übersicht und Arbeitsflüsse

Zusammenfassend sind nach der Einrichtung von OpenSSH gemäss Kapitel 4.2 folgende Funktionalitäten und Varianten implementiert:

1. Generierung von Schlüssel und Hinterlegung des Public-Keys auf dem Ziel-Host
   Kapitel 4.2.1
2. Erstellen einer SSH-Server-Grundkonfiguration zur Erfüllung des ermittelten Minimalstandards gemäss Kapitel 3.5.3
   Kapitel 4.2.2
3. Übernehmen der SSH-Server-Konfiguration sowie das Aktivieren und Starten des entsprechenden Serverdienstes `sshd`
   Kapitel 4.2.4
4. Implementation der Konfigurationen zu folgenden Anwendungsfällen, aufgeteilt auf unterschiedliche Benutzerkonten
   a) Kommandozeilenzugriff
      Kapitel 4.2.3
   b) Dateiübertragungen
      Kapitel 4.2.5
   c) Jumphost
      Kapitel 4.2.6
5. Verwendung des Authentisierungs-Agenten `ssh-agent` zur Hinterlegung von Private-Keys, um diese bei Agent Forwarding ggf. mit eingeschränkter Schlüssel-Nutzung weiterverwenden zu können und die zugehörige Passphrase nicht bei jedem SSH-Login eingeben zu müssen
   Kapitel 4.2.7
6. Zertifikate mittels Signierung von Public-Keys mit einer dedizierten CA erstellen und nutzen, womit eine zentrale Kontrolle über die zu verwendenden SSH-Schlüssel eingerichtet wird
   Kapitel 4.2.8
7. Verifikation von Fingerprints mittels DNS-SSHFP-Record
   Kapitel 4.2.9
8. Client-Verifikation mittels DNS-Adressabfragen und Aktivierung der DNS-Funktionalität in `sshd`
   Kapitel 4.2.10
9. FIDO2 Authentisierung mit einem YubiKey inkl. Zertifikat
   Kapitel 4.2.11

Da aus Zeitgründen u. a. gewisse Details bei Zertifikaten oder die Einschränkung der Key-Nutzung beim Authentisierungs-Agenten nicht tiefer betrachtet wurden, gibt es sicherlich weitere Möglichkeiten eine SSH-Server-Konfiguration weiter abzusichern. Die aufgeführten Varianten können zu unterschiedlichen SSH-Server-Konfigurationen führen, welche im Anhang in Kapitel B.1 hervorgehoben werden.

Mit zunehmender Sicherheit wächst die Komplexität des SSH-Servers, was mehr zu beachtende Faktoren mit sich bringt. Die nachfolgenden Tabellen listen Arbeitsflüsse mit Rücksichtnahme der aufgelisteten Varianten und zugehöriger Faktoren auf (wobei die Punkte der vorherigen Liste als jeweils zu beachtende Faktoren referenziert werden):

	Arbeitsfluss	
Fall	Grundkonfiguration	Zusätzlich bei Variante
Neuer Anwender	▶ Erstellung entsprechender Benutzer auf Client- und Server-Hosts ▶ Generierung dediziertes Schlüsselpaar mit Passphrase Listenpunkt 1 ▶ Hinterlegung des Public-Keys bei entsprechenden Benutzers auf den Server-Hosts Listenpunkt 1 ▶ Anpassung der Server-Konfiguration(en) mit `Match`-Bedingung für entsprechende Benutzer-, Gruppen-, Hostnamen- oder Adress-Muster und Anwendungsfall Listenpunkte 2, 4	▶ FIDO2 / YubiKey Listenpunkt 9 – Generierung des Schlüsselpaars mit YubiKey Listenpunkt 1 ▶ Zertifikate Listenpunkt 6 – Zertifikat zum Public-Key auf CA generieren und Anwender übergeben – Der Public-Key muss dann bei entsprechenden Ziel-Benutzer bzw. Server-Hosts nicht mehr hinterlegt werden, sofern sie den Zertifikaten der CA vertrauen ▶ Kombination der Varianten mit zuvor erwähnten Arbeitsflüssen Listenpunkte 6, 9

Tabelle 4.3.: Arbeitsflüsse für den Fall „Neuer Anwender“

	Arbeitsfluss	
Fall	Grundkonfiguration	Zusätzlich bei Variante
Neuer Server	▶ Erstellung entsprechender Benutzer auf Server-Hosts ▶ Server-Host-Keys beim Klonen eines vorhandenen Servers löschen und mit `ssh-keygen -A` neu erstellen Bei einer Neuinstallation ist wahrscheinlich ein Löschen nicht nötig, aber beim Neuerstellen des Servers durch Klonen eines bestehenden Hosts schon (im Falle von virtuellen Maschinen) ▶ Aufbau oder ggf. Kopie der SSH-Server-Konfiguration Listenpunkte 2, 3 ▶ Anwendung entsprechender Anwendungsfälle (Kommandozeilenzugriff, Dateiübertragungen, Jumphost), Forwarding/Weiterleitungen oder Agent Forwarding in der SSH-Server-Konfiguration Listenpunkte 2, 3, 4, 5	▶ DNS-Funktionalität Listenpunkt 8 – Mit der `UseDNS`-Option in der Server-Konfiguration das Verwenden von DNS-Domain-Namen in der Konfiguration ermöglichen – Logging bzw. Monitoring so einrichten, dass Clients, die im Server-Log auftauchen, weil ihre IP-Adresse nicht mit den entsprechenden DNS-Einträgen übereinstimmt, Warnungen oder Alarme auslösen ▶ DNS-SSHFP-Record Listenpunkt 7 – Fingerprints der Server-Host-Public-Keys als DNS-SSHFP-Records zur Verifikation durch die Clients zur Verfügung stellen und bestenfalls mit der Verwendung von DNSSEC absichern ▶ FIDO2 / YubiKey Listenpunkt 9 – Falls gewünscht und ausschliesslich FIDO Authenticators für den Server verwendet werden die benötigten Authentisierungsmethoden in der Server-Konfiguration anpassen, sodass nur Public-Key-Authentisierung notwendig ist (Multi-Faktor-Authentisierung durch FIDO Authenticator und PIN gegeben) Listenpunkte 2, 3 ▶ Zertifikate Listenpunkt 6 – Zertifikate zu den Host-Public-Keys auf CA generieren und bei Server hinterlegen – Der Public-Key muss dann nicht mehr bei den `known_hosts`-Datei auf dem Client hinterlegt werden, sofern dieser den Zertifikaten der CA vertraut ▶ Kombination der Varianten mit zuvor erwähnten Arbeitsflüssen Listenpunkte 6, 9, 7, 8

Tabelle 4.4.: Arbeitsflüsse für den Fall „Neuer Server“

	Arbeitsfluss	
Fall	Grundkonfiguration	Zusätzlich bei Variante
Verlust oder Kompromittierung eines Benutzer-Schlüssels	▶ Entfernen des entsprechenden Public-Keys bei entsprechenden Benutzern auf den Server-Hosts Listenpunkt 1 ▶ Vernichtung des Schlüssels, sofern vorhanden ▶ Logging bzw. Monitoring so einrichten, dass das Auftauchen des zugehörigen Fingerprints im Server-Log eine Warnung oder Alarm auslöst ▶ Vorgehen gemäss Fall „Neuer Anwender“ in Tabelle 4.3 ohne Erstellung weiterer Benutzer oder Anpassung der Server-Konfiguration(en)	▶ Wenn möglich „Key Revocation Lists“ mit `ssh-keygen` verwenden, um verteilte Schlüssel oder Zertifikate widerrufen zu können (im Rahmen dieser Arbeit nicht behandelt) [53] Wenn nicht möglich je nach Risiko des Keys weitere Massnahmen umsetzen (im schlimmsten Fall über die ganze Infrastruktur wenn möglich die Minimum-Schlüsselgrösse erhöhen oder die Verwendung des betroffenen Algorithmus unterbinden und betroffene Schlüssel ersetzen) ▶ FIDO2 / YubiKey Listenpunkt 9 – YubiKey zurücksetzen, sofern vorhanden Wenn nicht mehr vorhanden neuen YubiKey nehmen (Fall „Neuer Anwender“ in Tabelle 4.3)

Tabelle 4.5.: Arbeitsflüsse für den Fall „Verlust oder Kompromittierung eines Benutzer-Schlüssels“

	Arbeitsfluss	
Fall	Grundkonfiguration	Zusätzlich bei Variante
Verlust oder Kompromittierung eines Server-Schlüssels	▶ Entfernen des entsprechenden Public-Keys bei den `known_hosts`-Datei(en) auf den Clients Listenpunkt 5 ▶ Vernichtung des Schlüssels, sofern vorhanden ▶ Vorgehen gemäss Fall „Neuer Server“ in Tabelle 4.4 ohne Erstellung weiterer Benutzer oder Anpassung der Server-Konfiguration(en)	▶ Wenn möglich „Key Revocation Lists“ mit `ssh-keygen` verwenden, um verteilte Schlüssel oder Zertifikate widerrufen zu können (im Rahmen dieser Arbeit nicht behandelt) [53] Wenn nicht möglich je nach Risiko des Keys weitere Massnahmen umsetzen (im schlimmsten Fall über die ganze Infrastruktur wenn möglich die Minimum-Schlüsselgrösse erhöhen oder die Verwendung des betroffenen Algorithmus unterbinden und betroffene Schlüssel ersetzen) ▶ DNS-SSHFP-Record Listenpunkt 7 – DNS-SSHFP-Records mit Fingerprints des betroffenen Server-Host-Public-Keys durch neuen Fingerprint ersetzen und bestenfalls mit der Verwendung von DNSSEC absichern

Tabelle 4.6.: Arbeitsflüsse für den Fall „Verlust oder Kompromittierung eines Server-Schlüssels“

Fall	Arbeitsfluss
Ablauf eines Schlüssels oder Zertifikats	▶ Vorgehen gemäss Fall „Verlust oder Kompromittierung eines Benutzer-Schlüssels“ in Tabelle 4.5 oder „Verlust oder Kompromittierung eines Server-Schlüssels“ in Tabelle 4.6 (je nachdem ob es sich um einen Schlüssel oder Zertifikat für einen Benutzer oder Server handelt)

Tabelle 4.7.: Arbeitsflüsse für den Fall „Ablauf eines Schlüssels oder Zertifikats“

Fall	Arbeitsfluss
Regelmässige Durchführung	▶ Alle paar Jahre, z.B. alle 3 Jahre – Kompletter Austausch sämtlicher Schlüssel und Zertifikate inklusive CA (siehe Tabelle 4.7) ▶ Jährlich – Überprüfung des aktuellen Stands der Technik und Richtlinien, Abgleich mit Minimalstandard (Kapitel 3.5.3) und ggf. Anpassung der Server-Konfiguration Bei Verwendung nicht mehr empfohlener Algorithmen sind betroffene Schlüssel und Zertifikate zu ersetzen (siehe Tabelle 4.7) – Dokumentationen und Konzepte überprüfen und ggf. anpassen – Durchspielen von Szenarien (z.B. Schlüssel-Kompromittierung) – Funktionsprüfung von Logging und Monitoring – Sicherstellen der Server-Zugänge ohne SSH Je nachdem über andere Konsole oder Ansprechperson ▶ Monatlich oder öfter (bestenfalls automatisiert) – Aktualisierung der installierten Betriebssysteme inkl. Firmware und Software Bestenfalls mit automatisierten Funktionstests nach der Aktualisierung – Informieren über bekannte Schwachstellen – Prüfen des Monitorings – Sichern der Konfiguration und der Logs

Tabelle 4.8.: Arbeitsflüsse zur regelmässigen Durchführung

5. Verifikation

Zur Verifikation des Aufbaus und der Sicherstellung der Datenflüsse wird die Laborumgebung gemäss Kapitel 4.1 (inklusive Firewall-Konfiguration gemäss Abbildung 4.2) verwendet. Die Vorbedingungen der einzelnen Testfälle weisen die entsprechend verwendete Konfiguration auf.

Sofern es nicht anders erwähnt ist, werden die in den Tests verwendeten Server (`s1` und `s2`) mit der SSH-Server-Grundkonfiguration gemäss Kapitel 4.2.2 durchgeführt. Weitere Konfigurationen oder Konfigurationsanpassungen wie z.B. die Einrichtung für den Kommandozeilenzugriff gemäss Kapitel 4.2.3 werden ausdrücklich erwähnt.

Zudem wird für die Authentisierung beim SSH-Login unter dem Benutzer `user` auf dem Client-Host `c1` der in Kapitel 4.2.1 generierte Ed25519-Schlüssel verwendet, sofern der Test diesen benötigt. Beim Verwenden anderer Schlüssel wird dies explizit im Test aufgeführt.

Wird eine Anpassung an der SSH-Server-Konfiguration erwähnt, so wird impliziert, dass diese vor der Durchführung des nächsten Schritts entsprechend Kapitel 4.2.4 mit dem Befehl `doas rcctl reload sshd` neu eingelesen wurde.

Zur Übersicht folgt eine Auflistung der durchgeführten Testfälle in Kategorien aufgeteilt:

Testfälle

5.1. Allgemein

5.1.1. Kommunikation zwischen Client und Server

Mittels dieses Tests wird sichergestellt, dass die SSH-Verbindungen zwischen Client `c1` und Server `s1` erlaubt sind, wobei die Verbindung zwischen Client `c1` und Server `s2` blockiert wird.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet
  - Ed25519-Public-Key von `user@c1` unter `/home/jump/.ssh/authorized_keys` hinterlegt
  - Jumphost-Zugriff mittels Benutzer `jump` gemäss Kapitel 4.2.6, Quelltext 4.11 implementiert
- Server `s2`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet

#	Tätigkeit	Ergebnis
1	Auf `c1` wird mit folgendem Befehl eine SSH-Verbindung zu `s1` aufgebaut: `ssh cmd@192.168.1.1`	Der Fingerprint von `s1` wird angezeigt und ist zu bestätigen siehe Quelltext 5.1, Zeilen 2-5
2	Der Fingerprint von `s1` wird bestätigt	Fingerprint von `s1` ist für `user@c1` bestätigt siehe Quelltext 5.1, Zeilen 5-6
3	Der Login wird durch Eingabe der Passphrase des Keys und des Passworts des Benutzers `cmd` auf `s1` durchgeführt	Der Login wird erfolgreich durchgeführt siehe Quelltext 5.1, Zeilen 7-9
4	Die Sitzung wird durch Eingabe von `exit` beendet	Sitzung mit `s1` wird beendet siehe Quelltext 5.1, Zeilen 10-11
5	Auf `c1` wird mit folgendem Befehl eine SSH-Verbindung zu `s2` aufgebaut: `ssh cmd@192.168.2.1`	Es dauert circa 1 Minute bis der `ssh`-Befehl mit einer „Timeout“-Meldung den Verbindungsversuch abbricht siehe Quelltext 5.1, Zeilen 12-13
6	Im WebGUI der pfSense-Firewall wird unter „Status“ → „System Logs“ → „Firewall“ die Firewall-Log-Einträge nach Zeitpunkt absteigend sortiert angezeigt	Ein „Deny“-Eintrag von 192.168.100.1 (`c1`) zu 192.168.2.1 (`s2`) mit Zielport 22 (SSH) wird angezeigt siehe Abbildung 5.1

Tabelle 5.1.: Verifikation „Allgemein“ - „Kommunikation zwischen Client und Server“

```
c1$ ssh cmd@192.168.1.1
The authenticity of host '192.168.1.1 (192.168.1.1)' can't be established.
ED25519 key fingerprint is SHA256:ctWpch78E8V8qZEv74v72HQaHVTauCgB0xF5xF1OvRU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.1.1' (ED25519) to the list of known hosts.
Enter passphrase for key '/home/user/.ssh/id_ed25519':
cmd@192.168.1.1's password:
s1$
s1$ exit
Connection to 192.168.1.1 closed.
c1$ ssh cmd@192.168.2.1
ssh: connect to host 192.168.2.1 port 22: Operation timed out
```

Quelltext 5.1: Verifikation „Allgemein“ - „Kommunikation zwischen Client und Server“

Last 500 Firewall Log Entries. (Maximum 500)

Action	Time	Interface	Rule	Source	Destination
✖	Sep 10 09:54:49	C1	Default deny rule IPv4 (1000000103)	192.168.100.1:28145	192.168.2.1:22

Abbildung 5.1.: Firewall-Log-Eintrag, welcher das Blockieren der Kommunikation von 192.168.100.1 (`c1`) zu 192.168.2.1 (`s2`) mit Zielport 22 (SSH) angezeigt

Somit ist bewiesen, dass der Client `c1` nur direkt mit Server `s1` , aber nicht mit Server `s2` kommunizieren kann. Die Kommunikation zwischen den Servern `s1` und `s2` wird mittels Testfall u. a. in Kapitel 5.5 gezeigt.

Die Schritte zum Fingerprint können bei zukünftigen Testfällen ebenfalls vorkommen, sofern der Server-Host-Public-Key nicht auf dem Client unter `~/.ssh/known_hosts` eingetragen ist. Aus Platzgründen wird die Angabe und das Akzeptieren des Fingerprints nur noch erwähnt, wenn dies für den Testfall von Bedeutung ist.

5.1.2. Algorithmen-Wahl

Anhand dieses Tests wird stichprobenartig geprüft, ob andere als auf dem Server konfigurierte Algorithmen zugelassen werden.

Vorbedingungen:

- Client `c1`
  - CA-Public-Key gemäss Kapitel 4.2.8.6 im `known_hosts`-File hinterlegt
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub` sowie das Zertifikat `id_ed25519-cert.pub` (mit Principal `principalb`, siehe Kapitel 4.2.8.3)
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` hinterlegt
  - Datei `/home/cmd/.ssh/authorized_principals` gemäss Kapitel 4.2.8.2 angelegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet
  - Zertifikatsauthentisierung:
    - CA-Public-Key und `AuthorizedPrincipalsFile`-Option gemäss Kapitel 4.2.8.2 installiert und in Konfiguration hinterlegt
    - Zertifikatsauthentisierung ist nicht forciert (`AuthorizedKeysFile`-Wert entspricht der Grundkonfiguration in Kapitel 4.2.2)
    - Host-Zertifikate gemäss Kapitel 4.2.8.6 generiert und in Konfiguration hinterlegt

#	Tätigkeit	Ergebnis
1	Auf Server `s1` mit `sshd -G` die effektive Konfiguration ausgeben	Die jeweils verwendeten Algorithmen werden aufgelistet siehe Quelltext 5.2, Zeilen 3,4,6-10
2	Auf Client `c1` mit folgendem Befehl versuchen eine SSH-Verbindung zu `s1` mit einem nicht konfigurierten `cipher`-Wert aufzubauen: `ssh -c aes256-ctr cmd@192.168.1.1` Zur Erinnerung: Dem SSH-Client mögliche „Ciphers“ werden mit `ssh -Q cipher` abgefragt	Die Verbindung wird nicht aufgebaut mit der Meldung, dass keine gemeinsame „Cipher“ gefunden werden konnte siehe Quelltext 5.3
3	Auf Client `c1` mit folgendem Befehl versuchen eine SSH-Verbindung zu `s1` mit einem nicht konfigurierten `macs`-Wert aufzubauen: `ssh -v -m umac-128-etm@openssh.com \ cmd@192.168.1.1` Zur Erinnerung: Dem SSH-Client mögliche MACs werden mit `ssh -Q mac` abgefragt	Die Verbindung wird aufgebaut, jedoch nur weil der MAC des AES-GCM-„Ciphers“ verwendet wird siehe Quelltext 5.4, Zeilen 1-10 Die Konfiguration des `macs`-Werts hat nur bei „Ciphers“ ohne MAC einen Einfluss und würde bei einem MAC-Mismatch ähnlich wie zuvor bei den „Ciphers“ antworten (wurde ausserhalb dieser Arbeit verifiziert)

#	Tätigkeit	Ergebnis
4	Die Sitzung wird durch Eingabe von `exit` beendet	Sitzung mit `s1` wird beendet siehe Quelltext 5.4, Zeilen 11-12
5	Auf Client `c1` mit folgendem Befehl versuchen eine SSH-Verbindung zu `s1` mit einem nicht konfigurierten `KexAlgorithms`-Wert aufzubauen: `ssh -o "KexAlgorithms \ sntrup761x25519-sha512@openssh.com" \ cmd@192.168.1.1` Zur Erinnerung: Dem SSH-Client mögliche KEX-Algorithmen werden mit `ssh -Q kex` abgefragt	Die Verbindung wird nicht aufgebaut mit der Meldung, dass keine gemeinsame „key exchange method“ gefunden werden konnte siehe Quelltext 5.5
6	Auf Client `c1` mit folgendem Befehl versuchen eine SSH-Verbindung zu `s1` mit einem nicht konfigurierten `CASignatureAlgorithms`-Wert aufzubauen: `ssh -v -o "CASignatureAlgorithms \ ssh-dss" cmd@192.168.1.1` Zur Erinnerung: Dem SSH-Client mögliche Signatur-Algorithmen werden mit `ssh -Q sig` abgefragt	Die Verbindung wird aufgebaut mit der Meldung, das Host-Zertifikat mit einem unerlaubten Algorithmus signiert ist, somit keine passende CA findet, es jedoch mit dem unsignierten Schlüssel weiterversucht siehe Quelltext 5.6, Zeilen 1-15
7	Die Sitzung wird durch Eingabe von `exit` beendet	Sitzung mit `s1` wird beendet siehe Quelltext 5.6, Zeilen 16-17
8	Auf Client `c1` mit folgendem Befehl versuchen eine SSH-Verbindung zu `s1` mit einem nicht konfigurierten `HostKeyAlgorithms`-Wert aufzubauen: `ssh -o "HostKeyAlgorithms \ ssh-dss" cmd@192.168.1.1` Zur Erinnerung: Dem SSH-Client mögliche Schlüssel-Typen und Signatur-Algorithmen werden mit `ssh -Q key-sig` abgefragt	Die Verbindung wird nicht aufgebaut mit der Meldung, dass keine gemeinsamen Host-Key-Typen gefunden werden konnten siehe Quelltext 5.7
9	Auf Client `c1` mit folgendem Befehl versuchen eine SSH-Verbindung zu `s1` mit einem nicht konfigurierten `PubkeyAcceptedAlgorithms`-Wert aufzubauen: `ssh -v -o "PubkeyAcceptedAlgorithms \ ssh-dss" cmd@192.168.1.1` Zur Erinnerung: Dem SSH-Client mögliche Schlüssel-Typen und Signatur-Algorithmen werden mit `ssh -Q key-sig` abgefragt	Die Verbindung wird nicht aufgebaut mit der Meldung, dass die Verbindung aufgrund des Public-Keys verweigert wurde In der Ausgabe ist ersichtlich, dass der vorhandene Key nicht verwendet wurde und somit kein Schlüssel für die Public-Key-Authentisierung versucht wurde siehe Quelltext 5.8

Tabelle 5.2.: Verifikation „Allgemein“ - „Algorithmen-Wahl“

Hiermit wird aufgezeigt, dass die auf dem Server hinterlegten Algorithmen forciert werden und keine anderen Werte zulassen. Zusätzlich sind folgende Erkenntnisse zum Verhalten des SSH-Servers hervorzuheben:

- Beinhaltet der in `Cipher` hinterlegte Algorithmus einen MAC, so wird dieser verwendet und die Werte unter `MACs` in der Konfiguration ignoriert
- Wird ein unpassender `CASignatureAlgorithms`-Wert definiert, wird nicht mehr das Server-Host-Zertifkat, sondern der Host-Schlüssel direkt angeschaut (hier gilt es dann eine passende `HostKeyAlgorithms`-Einstellung zu haben)

```
s1$ sshd -G
...
ciphers aes128-gcm@openssh.com,aes256-gcm@openssh.com
macs hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha2-256,hmac-sha2-512
...
kexalgorithms diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521
casignaturealgorithms ssh-ed25519,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,sk-ssh-ed25519@openssh.com,sk-ecdsa-sha2-nistp256@openssh.com,rsa-sha2-512,rsa-sha2-256
...
hostkeyalgorithms ssh-ed25519-cert-v01@openssh.com,ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com,sk-ssh-ed25519-cert-v01@openssh.com,sk-ecdsa-sha2-nistp256-cert-v01@openssh.com,rsa-sha2-512-cert-v01@openssh.com,rsa-sha2-256-cert-v01@openssh.com,ssh-ed25519,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,sk-ssh-ed25519@openssh.com,sk-ecdsa-sha2-nistp256@openssh.com,rsa-sha2-512,rsa-sha2-256
pubkeyacceptedalgorithms ssh-ed25519-cert-v01@openssh.com,ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com,sk-ssh-ed25519-cert-v01@openssh.com,sk-ecdsa-sha2-nistp256-cert-v01@openssh.com,rsa-sha2-512-cert-v01@openssh.com,rsa-sha2-256-cert-v01@openssh.com,ssh-ed25519,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,sk-ssh-ed25519@openssh.com,sk-ecdsa-sha2-nistp256@openssh.com,rsa-sha2-512,rsa-sha2-256
...
```

Quelltext 5.2: Verifikation „Allgemein“ - „Algorithmen-Wahl“ - Algorithmen auf `s1`

```
c1$ ssh -c aes256-ctr cmd@192.168.1.1
Unable to negotiate with 192.168.1.1 port 22: no matching cipher found. Their offer: aes128-gcm@openssh.com,aes256-gcm@openssh.com
```

Quelltext 5.3: Verifikation „Allgemein“ - „Algorithmen-Wahl“ - Verbindungsversuch mit anderer Cipher von `c1` zu `s1`

```
c1$ ssh -v -m umac-128-etm@openssh.com cmd@192.168.1.1
...
debug1: kex: server->client cipher: aes128-gcm@openssh.com MAC: <implicit>
    compression: none
debug1: kex: client->server cipher: aes128-gcm@openssh.com MAC: <implicit>
    compression: none
...
Enter passphrase for key '/home/user/.ssh/id_ed25519':
...
cmd@192.168.1.1's password:
...
s1$
s1$ exit
c1$
```

Quelltext 5.4: Verifikation „Allgemein“ - „Algorithmen-Wahl“ - Verbindungsversuch mit anderer MAC von `c1` zu `s1`

```
c1$ ssh -o "KexAlgorithms \
> sntrup761x25519-sha512@openssh.com" \
> cmd@192.168.1.1
Unable to negotiate with 192.168.1.1 port 22: no matching key exchange method
    found. Their offer: diffie-hellman-group16-sha512,diffie-hellman-group18-
    sha512,curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,
    ecdh-sha2-nistp384,ecdh-sha2-nistp521
```

Quelltext 5.5: Verifikation „Allgemein“ - „Algorithmen-Wahl“ - Verbindungsversuch mit anderem KEX-Algorithmus von `c1` zu `s1`

```
c1$ ssh -v -o "CASignatureAlgorithms ssh-dss" cmd@192.168.1.1
...
debug1: Server host certificate: ssh-ed25519-cert-v01@openssh.com SHA256:
    ctWpch78E8V8qZEv74v72HQaHVTauCgB0xF5xF1OvRU, serial 0 ID "host_s1" CA ssh-
    ed25519 SHA256:GwSQXlRbLXCg51XIVqJatsqKZL8JZxr91tBNRU5DS3Q valid forever
...
debug1: Host '192.168.1.1' is known and matches the ED25519-CERT host
    certificate.
debug1: Found CA key in /home/user/.ssh/known_hosts:1
Certificate signed with disallowed algorithm
debug1: No matching CA found. Retry with plain key
debug1: Host '192.168.1.1' is known and matches the ED25519 host key.
...
Enter passphrase for key '/home/user/.ssh/id_ed25519':
...
cmd@192.168.1.1's password:
...
s1$
s1$ exit
c1$
```

Quelltext 5.6: Verifikation „Allgemein“ - „Algorithmen-Wahl“ - Verbindungsversuch mit anderem CA-Signatur-Algorithmus von `c1` zu `s1`

```
c1$ ssh -o "HostKeyAlgorithms ssh-dss" cmd@192.168.1.1
Unable to negotiate with 192.168.1.1 port 22: no matching host key type found.
     Their offer: rsa-sha2-512,rsa-sha2-256,rsa-sha2-512-cert-v01@openssh.com,
    rsa-sha2-256-cert-v01@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp256-
    cert-v01@openssh.com,ssh-ed25519,ssh-ed25519-cert-v01@openssh.com
```

Quelltext 5.7: Verifikation „Allgemein“ - „Algorithmen-Wahl“ - Verbindungsversuch mit anderem Host-Key-Typen von `c1` zu `s1`

```
c1$ ssh -v -o "PubkeyAcceptedAlgorithms ssh-dss" cmd@192.168.1.1
O...
debug1: Skipping ssh-ed25519 key /home/user/.ssh/id_ed25519 - corresponding
   algo not in PubkeyAcceptedAlgorithms
debug1: Skipping ssh-ed25519-cert-v01@openssh.com key /home/user/.ssh/
   id_ed25519 - corresponding algo not in PubkeyAcceptedAlgorithms
debug1: Will attempt key: /home/user/.ssh/id_rsa
debug1: Will attempt key: /home/user/.ssh/id_ecdsa
debug1: Will attempt key: /home/user/.ssh/id_ecdsa_sk
debug1: Will attempt key: /home/user/.ssh/id_ed25519_sk
debug1: Will attempt key: /home/user/.ssh/id_xmss
debug1: Will attempt key: /home/user/.ssh/id_dsa
debug1: SSH2_MSG_EXT_INFO received
debug1: kex_input_ext_info: server-sig-algs=<ssh-ed25519,sk-ssh-
   ed25519@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-
   nistp521,sk-ecdsa-sha2-nistp256@openssh.com,webauthn-sk-ecdsa-sha2-
   nistp256@openssh.com,ssh-dss,ssh-rsa,rsa-sha2-256,rsa-sha2-512>
debug1: kex_input_ext_info: publickey-hostbound@openssh.com=<0>
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Trying private key: /home/user/.ssh/id_rsa
debug1: Trying private key: /home/user/.ssh/id_ecdsa
debug1: Trying private key: /home/user/.ssh/id_ecdsa_sk
debug1: Trying private key: /home/user/.ssh/id_ed25519_sk
debug1: Trying private key: /home/user/.ssh/id_xmss
debug1: Trying private key: /home/user/.ssh/id_dsa
debug1: No more authentication methods to try.
cmd@192.168.1.1: Permission denied (publickey).
```

Quelltext 5.8: Verifikation „Allgemein“ - „Algorithmen-Wahl“ - Verbindungsversuch mit anderem Public-Key-Typen von `c1` zu `s1`

5.1.3. Login mit Benutzer „root“

Dieser Testfall zeigt auf, dass der Zugang mit Benutzer `root` nur mit entsprechenden Konfigurationsanpassungen funktioniert und gemäss der definierten Grundkonfiguration nicht zugelassen wird.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` hinterlegt
  - Ed25519-Public-Key von `user@c1` unter `/root/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet
  - Kommandozeilenzugriff für Benutzer `root` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet

#	Tätigkeit	Ergebnis
1	Auf Client `c1` mit folgendem Befehl als Benutzer `root` auf dem Server `s1` einloggen: `ssh root@192.168.1.1`	Der Login als Benutzer `root` wird mit der Meldung `Permission denied (publickey)` abgelehnt
2	Auf Server `s1` in der SSH-Serverkonfiguration den Wert `PermitRootLogin` wie folgt anpassen: `PermitRootLogin prohibit-password` Zusätzlich wird die Zeile mit `AllowGroups` entfernt	Die Konfiguration wird übernommen
3	Auf Client `c1` mit folgendem Befehl als Benutzer `root` auf dem Server `s1` einzuloggen: `ssh root@192.168.1.1`	Aufgrund dass in der Grundkonfiguration nach dem Public-Key und Passwort gefragt wird, kann der Login nach Eingabe der Key-Passphrase und darauffolgender Eingabe des korrekten Passworts von `root` nicht erfolgen (es wird nur immer wieder nach dem Passwort gefragt, aber keines akzeptiert)
4	Auf Server `s1` in der SSH-Serverkonfiguration den Wert `PermitRootLogin` wie folgt anpassen: `PermitRootLogin yes`	Die Konfiguration wird übernommen
5	Auf Client `c1` mit folgendem Befehl als Benutzer `root` auf dem Server `s1` einzuloggen: `ssh root@192.168.1.1`	Der Login als Benutzer `root` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich

Tabelle 5.3.: Verifikation „Allgemein“ - „Login mit Benutzer „root““

5.1.4. Unzulässiger Benutzer

Anhand dieses Testfalls wird geprüft, ob ein Login mit einem Benutzer, welcher nicht in der Gruppe `sshaccess` auf dem Server hinterlegt ist, erfolgen kann.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet

#	Tätigkeit	Ergebnis
1	Auf Server `s1` mit dem Befehl `id cmd` Informationen zum Benutzer `cmd` ausgeben	Die Gruppenzugehörigkeiten von `cmd` werden ausgegeben: Gruppe `cmd` und Gruppe `sshaccess`
2	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich
3	Die Sitzung durch Eingabe von `exit` beenden	Sitzung mit `s1` wird beendet

4	Auf Server `s1` als privilegierter Benutzer in Datei `/etc/group` folgende Zeile anpassen:
	`sshaccess:*:1005:cmd,file,jump,agent`
	zu
	`sshaccess:*:1005:file,jump,agent`
		Benutzer `cmd` wurde aus der Gruppe `sshaccess` entfernt

5	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh cmd@192.168.1.1`	Der Login als Benutzer `cmd` wird mit der Meldung `Permission denied (publickey)` abgelehnt
6	Auf Server `s1` als berechtigter Benutzer die letzten Zeilen der Logdatei `/var/log/authlog` betrachten	Das Log zeigt auf, dass der Login aufgrund fehlender Gruppenzugehörigkeit nicht erlaubt ist siehe Quelltext 5.9

Tabelle 5.4.: Verifikation „Allgemein“ - „Unzulässiger Benutzer“

```
1 Sep 10 14:37:23 s1 sshd[29533]: User cmd from 192.168.100.1 not allowed
     because none of user's groups are listed in AllowGroups
2 Sep 10 14:37:23 s1 sshd[29533]: Connection closed by invalid user cmd
     192.168.100.1 port 1800 [preauth]
```

Quelltext 5.9: Verifikation „Allgemein“ - „Unzulässiger Benutzer“ - Ausgabe `/var/log/authlog` auf dem Server `s1`

Der Benutzer `cmd` wird nach diesem Testfall wieder der Gruppe `sshaccess` zugewiesen (siehe Kapitel 4.2).

5.1.5. Automatisches Schliessen nicht-authentisierter Verbindungen

Anhand dieses Testfalls wird aufgezeigt, dass nicht-authentisierte Verbindungen nach der definierten Dauer geschlossen werden.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet

#	Tätigkeit	Ergebnis
1	Auf Client `c1` mit folgendem Befehl eine SSH-Verbindung als Benutzer `cmd` auf dem Server `s1` initiieren, aber nicht einloggen: `ssh cmd@192.168.1.1`	Die Abfrage nach der Key-Passphrase bleibt unbeantwortet
2	Über 1 Minute warten (länger als die `LoginGraceTime` der SSH-Serverkonfiguration)	Die Abfrage nach der Key-Passphrase wird immer noch angezeigt
3	Key-Passphrase eingeben	Die Verbindung wird vom Server geschlossen
4	Auf Server `s1` als berechtigter Benutzer die letzten Zeilen der Logdatei `/var/log/authlog` betrachten	Das Log zeigt auf, dass ein Timeout vor der Authentisierung stattfand siehe Quelltext 5.10

Tabelle 5.5.: Verifikation „Allgemein“ - „Automatisches Schliessen nicht-authentisierter Verbindungen“

```
1 Sep 10 14:50:31 s1 sshd[12684]: fatal: Timeout before authentication for
     192.168.100.1 port 18143
```

Quelltext 5.10: Verifikation „Allgemein“ - „Automatisches Schliessen nicht-authentisierter Verbindungen“ - Ausgabe `/var/log/authlog` auf dem Server `s1`

5.1.6. TCP-Forwarding

Dieser Testfall prüft das TCP-Forwarding (Local und Remote Forwarding). Der Authentisierungs-Agent inklusive dessen Forwarding wird in Kapitel 5.6 geprüft. Das Forwarding von Unix Sockets wird aus Zeitgründen nicht geprüft. Das X11-Forwarding wird aufgrund fehlender graphischer Oberfläche nicht geprüft [74].

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/jump/.ssh/authorized_keys` hinterlegt
  - Die Jumphost-Konfiguration für Benutzer `jump` gemäss Quelltext 5.11 eingerichtet (am Ende der SSH-Server-Konfiguration angefügt)

```
# User jump ----------------------------------------------------------------
Match User jump
    ForceCommand               none
```

Quelltext 5.11: Verifikation „Allgemein“ - „TCP-Forwarding“ - Konfiguration für Benutzer `jump` auf dem Server `s1`

Local Forwarding

#	Tätigkeit	Ergebnis

1	Auf Client `c1` mit folgendem Befehl ein Local Forwarding mit Benutzer `cmd` auf dem Server `s1` anfragen und einloggen:
	`ssh -v -L 11443:www.openbsd.com:443 \`
	` jump@192.168.1.1`
		Der Benutzer `jump` ist eingeloggt

2	Auf Client `c1` neben der aktiven SSH-Session in einer weiteren Shell mit folgendem Befehl prüfen, ob auf `c1` Ports für das Local Forwarding geöffnet wurden: `netstat -an | grep 11443`	Der Port 11443 ist auf der Localhost-Adresse via IPv4 und IPv6 geöffnet

3	Auf Client `c1` neben der aktiven SSH-Session versuchen auf den Port zuzugreifen (z.B. durch Abfrage des Zertifikats auf dem Port):
	`openssl s_client \`
	` -connect 127.0.0.1:11443`
		Der Zugriff ist erfolglos, es wird kein Zertifikat ausgegeben
		In der aktiven SSH-Session wird ausgegeben, dass die Forwarding-Verbindung nicht geöffnet wurde siehe Quelltext 5.12
		Zudem ist in `/var/log/authlog` folgender Eintrag vorzufinden:
		`refused local port forward:`
		` originator 127.0.0.1 port 9355,`
		` target www.openbsd.com port 443`

4	Aktive SSH-Session mittels Tastenkombination „Ctrl+C“ schliessen	Die Session ist geschlossen

[74] Es wurden keine X11-Komponenten in der Laborumgebung installiert

#	Tätigkeit	Ergebnis

5	Auf Server `s1` in der SSH-Serverkonfiguration für Benutzer `jump` folgende Werte hinzufügen:
	`DisableForwarding no`
	`AllowTcpForwarding yes`
	`PermitOpen www.openbsd.com:443`
		Die Konfiguration wird übernommen

6	Auf Client `c1` mit folgendem Befehl ein Local Forwarding mit Benutzer `cmd` auf dem Server `s1` anfragen und einloggen:
	`ssh -v -L 11443:www.openbsd.com:443 \`
	`jump@192.168.1.1`
		Der Benutzer `jump` ist eingeloggt

7	Auf Client `c1` neben der aktiven SSH-Session versuchen auf den Port zuzugreifen (z.B. durch Abfrage des Zertifikats auf dem Port):
	`openssl s_client \`
	`-connect 127.0.0.1:11443`
		Der Zugriff ist erfolgreich, es wird das Zertifikat von www.openbsd.com unter Port 443 ausgegeben
		In der aktiven SSH-Session wird die Forwarding-Verbindung ohne Fehler aufgelistet
		siehe Quelltext 5.13

8	Aktive SSH-Session mittels Tastenkombination „Ctrl+C“ schliessen	Die Session ist geschlossen
9	Auf Server `s1` wird die SSH-Serverkonfiguration für Benutzer `jump` entsprechend den Vorbedingungen wiederhergestellt	Die Konfiguration wird übernommen

Tabelle 5.6.: Verifikation „Allgemein“ - „TCP-Forwarding“ - Local Forwarding

```
1 debug1: Connection to port 11443 forwarding to www.openbsd.com port 443
     requested.
2 debug1: channel 3: new direct-tcpip [direct-tcpip] (inactive timeout: 0)
3 channel 3: open failed: administratively prohibited: open failed
4 debug1: channel 3: free: direct-tcpip: listening port 11443 for www.openbsd.
     com port 443, connect from 127.0.0.1 port 1829 to 127.0.0.1 port 11443,
     nchannels 4
```

Quelltext 5.12: Verifikation „Allgemein“ - „TCP-Forwarding“ - Local Forwarding SSH-Client-Ausgabe ohne zusätzliche Konfiguration

```
1 debug1: Connection to port 11443 forwarding to www.openbsd.com port 443
     requested.
2 debug1: channel 3: new direct-tcpip [direct-tcpip] (inactive timeout: 0)
```

Quelltext 5.13: Verifikation „Allgemein“ - „TCP-Forwarding“ - Local Forwarding SSH-Client-Ausgabe mit Konfiguration

Remote Forwarding

#	Tätigkeit	Ergebnis

1	Auf Client `c1` mit folgendem Befehl ein Remote Forwarding mit Benutzer `cmd` auf dem Server `s1` anfragen und einloggen:
	`ssh -R 8080:www.openbsd.com:443 \`
	`jump@192.168.1.1`
		Der Benutzer `jump` ist eingeloggt, jedoch wird ausgegeben, dass das Remote Forwarding fehlgeschlagen ist
		siehe Quelltext 5.14

2	Auf Server `s1` neben der aktiven SSH-Session in einer weiteren Shell mit folgendem Befehl prüfen, ob auf `s1` Ports für das Remote Forwarding geöffnet wurden: `netstat -an | grep 8080`	Der Port 8080 ist nicht geöffnet
3	Aktive SSH-Session mittels Tastenkombination „Ctrl+C“ schliessen	Die Session ist geschlossen

4	Auf Server `s1` in der SSH-Serverkonfiguration für Benutzer `jump` folgende Werte hinzufügen:
	`DisableForwarding no`
	`AllowTcpForwarding yes`
	`PermitListen 8080`
		Die Konfiguration wird übernommen

5	Auf Client `c1` mit folgendem Befehl ein Remote Forwarding mit Benutzer `cmd` auf dem Server `s1` anfragen und einloggen:
	`ssh -R 8080:www.openbsd.com:443 \`
	`jump@192.168.1.1`
		Der Benutzer `jump` ist eingeloggt

6	Auf Server `s1` neben der aktiven SSH-Session in einer weiteren Shell mit folgendem Befehl prüfen, ob auf `s1` Ports für das Remote Forwarding geöffnet wurden: `netstat -an | grep 8080`	Der Port 8080 ist auf der Localhost-Adresse via IPv4 und IPv6 geöffnet

7	Auf Server `s1` neben der aktiven SSH-Session versuchen auf den Port zuzugreifen (z.B. durch Abfrage des Zertifikats auf dem Port):
	`openssl s_client \`
	`-connect 127.0.0.1:8080`
		Der Zugriff ist erfolgreich, es wird das Zertifikat von www.openbsd.com unter Port 443 ausgegeben
		In der aktiven SSH-Session wird zu dieser Forwarding-Verbindung nichts ausgegeben

8	Aktive SSH-Session mittels Tastenkombination „Ctrl+C“ schliessen	Die Session ist geschlossen

Tabelle 5.7.: Verifikation „Allgemein“ - „TCP-Forwarding“ - Remote Forwarding

```
1 Warning: remote port forwarding failed for listen port 8080
```

Quelltext 5.14: Verifikation „Allgemein“ - „TCP-Forwarding“ - Remote Forwarding SSH-Client-Ausgabe ohne zusätzliche Konfiguration

5.2. Kommandozeilenzugriff

5.2.1. Konfiguration

Anhand dieses Testfalls wird das Verhalten der Konfiguration für Kommandozeilenzugriff betrachtet.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 nicht eingerichtet

#	Tätigkeit	Ergebnis
1	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich, wird jedoch gleich wieder getrennt Zudem wird folgende Information eingeblendet: `PTY allocation request failed ...`

2	Auf Server `s1` in der SSH-Serverkonfiguration für Benutzer `cmd` werden folgende Werte angefügt:
	`Match User cmd`
	`    PermitTTY yes`
		Die Konfiguration wird übernommen

3	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich, wird jedoch gleich wieder getrennt Die Information zur „PTY allocation“ wird nicht mehr angezeigt
4	Auf Server `s1` in der SSH-Serverkonfiguration für Benutzer `cmd` werden folgende Werte angefügt: `    ForceCommand none`	Die Konfiguration wird übernommen
5	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich Es steht eine Shell zur Verfügung, mit welcher Befehle als Benutzer `cmd` auf dem Server `s1` ausgeführt werden können

Tabelle 5.8.: Verifikation „Kommandozeilenzugriff“ - „Konfiguration“

5.2.2. Dateiübertragungen bei Kommandozeilenzugriff

Gemäss Kapitel 4.2.3 ist es aufgrund der `Subsystem`-Definition in der Grundkonfiguration möglich, mit Kommandozeilen-Benutzer wie z.B. `cmd` im letzten Testfall Dateiübertragungen mittels SFTP durchzuführen. Dieser Testfall zeigt mögliche Dateiübertragungen mit und ohne SFTP-Subsystem auf.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet
  - Testarchiv für Dateiübertragungen unter `/tmp/test.tar.gz`, in diesem Testfall beispielsweise erstellt mittels folgendem Befehl:
    `tar cvfz /tmp/test.tar.gz /bsd /etc`
    SHA-256-Hash: `6dca22094d15b7fc2e6ef76341568f9982478c45c59fee9fdc88f33c4c4b98e5`

#	Tätigkeit	Ergebnis

1	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` das Testarchiv vom Server `s1` in das derzeitige Arbeitsverzeichnis kopieren:
	`scp \`
	`cmd@192.168.1.1:/tmp/test.tar.gz .`
		Der Kopiervorgang ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich

2	Auf Client `c1` mit folgendem Befehl den SHA-256-Hash der Datei ausgeben und kontrollieren: `sha256 test.tar.gz`	Der SHA-256-Hash entspricht der Datei `/tmp/test.tar.gz` auf `s1` Der Kopiervorgang wurde erfolgreich durchgeführt
3	Auf Client `c1` mit folgendem Befehl die Testdatei löschen: `rm test.tar.gz`	Die Testdatei wurde auf `c1` gelöscht
4	Auf Server `s1` in der SSH-Serverkonfiguration die Zeile mit `Subsystem` entfernen	Die Konfiguration wird übernommen

5	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` das Testarchiv vom Server `s1` in das derzeitige Arbeitsverzeichnis kopieren:
	`scp \`
	`cmd@192.168.1.1:/tmp/test.tar.gz .`
		Der Kopiervorgang ist nach Eingabe der Key-Passphrase und des Benutzerkennworts nicht erfolgreich, die Verbindung wird mit u. a. folgender Ausgabe geschlossen:
		`subsystem request failed ...`

6	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` das Testarchiv vom Server `s1` in das derzeitige Arbeitsverzeichnis kopieren:
	`ssh cmd@192.168.1.1 \`
	`cat /tmp/test.tar.gz \`
	`> test.tar.gz`
		Der „Kopiervorgang“ ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich
		Hiermit wird die Datei vom Server ausgegeben und die Ausgabe in eine Datei auf dem Client weitergeleitet

7	Auf Client `c1` mit folgendem Befehl den SHA-256-Hash der Datei ausgeben und kontrollieren: `sha256 test.tar.gz`	Der SHA-256-Hash entspricht der Datei `/tmp/test.tar.gz` auf `s1` Der Kopiervorgang wurde erfolgreich durchgeführt
8	Auf Client `c1` mit folgendem Befehl die Testdatei löschen: `rm test.tar.gz`	Die Testdatei wurde auf `c1` gelöscht

Tabelle 5.9.: Verifikation „Kommandozeilenzugriff“ - „Dateiübertragungen bei Kommandozeilenzugriff“

Somit ist bewiesen, dass Benutzer mit Kommandozeilenzugriff mit und ohne Anwendung von SFTP Dateien kopieren können. Ohne SFTP (mittels Anwendung `ssh`) können einzelne Dateien mittels `cat` ausgegeben und diese Ausgabe in eine lokale Datei umgeleitet werden. Um Ordner zu kopieren, könnten hierbei die entsprechenden Ordner zuerst in eine Archivdatei gepackt und dann als eine Datei übertragen werden. Dieser Testfall zeigt zwar lediglich das Kopieren vom Server zum Client, jedoch wird in Kapitel 4.2.1, Quelltext 4.3 das „Kopieren“ vom Client zum Server ohne SFTP demonstriert [75].

[75] Dort werden Daten an die bestehende Zieldatei angefügt, jedoch ist ein Überschreiben der Zieldatei mit `>` anstelle von `>>` auch möglich

5.2.3. Einschränkung auszuführender Befehle

Dieser Testfall prüft die Option `ForceCommand` der SSH-Serverkonfiguration. Hierbei wird eingeschränkt, welche Befehle ein Benutzer mit Kommandozeilenzugriff ausführen kann.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` hinterlegt
  - Skript gemäss Kapitel 4.2.3, Quelltext 4.6 unter `/home/cmd/selection.sh` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.7 eingerichtet
  - Testarchiv für Dateiübertragungen unter `/tmp/test.tar.gz`, siehe vorheriger Testfall

#	Tätigkeit	Ergebnis
1	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh cmd@192.168.1.1`	Der Login ist erfolgreich, das hinterlegte Skript wird ausgeführt und die SSH-Sitzung bei Skript-Ende geschlossen
2	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen und einen anderen Befehl ausführen (z.B. Auflisten des Verzeichnisses `/tmp`): `ssh cmd@192.168.1.1 ls /tmp`	Das Ergebnis entspricht demselben des vorherigen Schrittes Der `ls`-Befehl wird nicht ausgeführt

3	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` das Testarchiv vom Server `s1` in das derzeitige Arbeitsverzeichnis kopieren:
	`scp \`
	`cmd@192.168.1.1:/tmp/test.tar.gz .`
		Der Kopiervorgang wird nach Eingabe der Key-Passphrase und des Benutzerkennworts nicht ausgeführt
		siehe Quelltext 5.15

4	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf Server `s1` eine interaktive SFTP-Sitzung starten: `sftp cmd@192.168.1.1`	Das Ergebnis entspricht demselben des vorherigen Schrittes

Tabelle 5.10.: Verifikation „Kommandozeilenzugriff“ - „Einschränkung auszuführender Befehle“

```
1 c1$ scp cmd@192.168.1.1:/tmp/test.tar.gz .
2 Enter passphrase for key '/home/user/.ssh/id_ed25519':
3 cmd@192.168.1.1's password:
4 scp: Received message too long 1164866661
5 scp: Ensure the remote shell produces no output for non-interactive sessions.
```

Quelltext 5.15: Verifikation „Kommandozeilenzugriff“ - „Einschränkung auszuführender Befehle“ - Dateiübertragungsversuch mit definierter `ForceCommand`-Option

Hiermit wird gezeigt, dass mittels `ForceCommand` die auszuführenden Befehle und Dateiübertragungen für Benutzer mit Kommandozeilenzugriff eingeschränkt bzw. unterbunden werden können.

5.3. Dateiübertragungen

5.3.1. Einschränkung der Zielverzeichnisse

Dieser Testfall betrachtet Dateiübertragungen mit eingeschränkten Zielverzeichnissen (SSH-Server-Option `ChrootDirectory`, siehe Kapitel 4.2.5). Ob auf dem Client als Applikation `scp` oder `sftp` verwendet wird, spielt hierbei keine Rolle. Zur Demonstration beide Applikationen verwendet.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
  - Testarchiv für Dateiübertragungen unter `/tmp/file.tar.gz`, in diesem Testfall beispielsweise erstellt mittels folgendem Befehl:
    `tar cvfz /tmp/file.tar.gz /bsd /etc`
    SHA-256-Hash: `84371461d781f2ef2668bffea46ed2a19c5009b2e9c21a3c4ea54163a8bbf750`
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/file/.ssh/authorized_keys` hinterlegt
  - Dateiübertragung für Benutzer `file` gemäss Quelltext 5.16 eingerichtet (am Ende der SSH-Server-Konfiguration angefügt)

```
1 # User file ------------------------------------------------------------------
2 Match User file
3     ForceCommand               none
```

Quelltext 5.16: Verifikation „Dateiübertragungen“ - „Einschränkung der Zielverzeichnisse“ - Konfiguration für Benutzer `file`

#	Tätigkeit	Ergebnis
1	Auf Client `c1` mit folgendem Befehl als Benutzer `file` auf Server `s1` eine interaktive SFTP-Sitzung starten: `sftp file@192.168.1.1`	Die interaktive SFTP-Sitzung steht nach Eingabe der Key-Passphrase und des Benutzerkennworts zur Verfügung
2	Mittels folgendem Befehl die Datei `/tmp/file.tar.gz` vom Client auf den Server hochladen: `put /tmp/file.tar.gz`	Die Datei wird nach `/home/file/file.tar.gz` hochgeladen
3	Mittels folgendem Befehl die Datei wieder vom Server löschen: `rm file.tar.gz`	Die Datei wurde gelöscht

4	Mittels folgendem Befehl die Datei `/tmp/file.tar.gz` vom Client auf den Server erneut hochladen:
	`scp \`
	` /tmp/file.tar.gz file@192.168.1.1:`
		Die Datei wird nach Eingabe der Key-Passphrase und des Benutzerkennworts nach `/home/file/file.tar.gz` hochgeladen

#	Tätigkeit	Ergebnis

5	Auf Server `s1` als Benutzer `root` (oder mittels `doas`-Befehl) folgende Verzeichnisse anlegen:
	`mkdir -p /data/test/download`
	`mkdir -p /data/test/upload`
	`chown file /data/test/upload`
		Die Verzeichnisse wurden angelegt, wobei der Benutzer `file` Besitzer des Verzeichnisses `/data/test/upload` ist und dorthin schreiben kann

6	Auf Server `s1` die SSH-Serverkonfiguration für Benutzer `file` entsprechend Quelltext 5.17 konfigurieren	Die Konfiguration wird übernommen
7	Auf Client `c1` mit folgendem Befehl als Benutzer `file` auf Server `s1` eine interaktive SFTP-Sitzung starten: `sftp file@192.168.1.1`	Die interaktive SFTP-Sitzung steht nach Eingabe der Key-Passphrase und des Benutzerkennworts zur Verfügung
8	Mittels folgendem Befehl die Datei `/tmp/file.tar.gz` vom Client auf den Server hochladen: `put /tmp/file.tar.gz`	Der Upload wird mit der Meldung `Permission denied` verweigert
9	Mittels folgendem Befehl die verfügbaren Ordner und Pfade anzeigen: `ls`	Die zuvor erstellten Ordner `download` und `upload` werden angezeigt
10	Mittels folgendem Befehl in den Ordner `upload` wechseln: `cd upload`	Verzeichniswechsel durchgeführt
11	Mittels folgendem Befehl die Datei `/tmp/file.tar.gz` vom Client auf den Server hochladen: `put /tmp/file.tar.gz`	Die Datei wird aus Client-Sicht nach `/upload/file.tar.gz` hochgeladen
12	Auf Server `s1` mit folgendem Befehl den SHA-256-Hash der Datei ausgeben und kontrollieren: `sha256 /data/test/upload/file.tar.gz`	Der SHA-256-Hash entspricht der Datei `/tmp/file.tar.gz` auf `c1` Der Kopiervorgang wurde erfolgreich durchgeführt

Tabelle 5.11.: Verifikation „Dateiübertragungen“ - „Einschränkung der Zielverzeichnisse“

```
# User file ------------------------------------------------------------------
Match User file
        ForceCommand              internal-sftp
        ChrootDirectory           /data/test
```

Quelltext 5.17: Verifikation „Dateiübertragungen“ - „Einschränkung der Zielverzeichnisse“ - Konfiguration für Benutzer `file`

5.3.2. Kommandozeilenzugriff bei Dateiübertragungen

Mit diesem Testfall wird der Kommandozeilenzugriff bei Dateiübertragungen getestet.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/file/.ssh/authorized_keys` hinterlegt
  - Dateiübertragung für Benutzer `file` gemäss Quelltext 5.18 eingerichtet (am Ende der SSH-Server-Konfiguration angefügt)

```
# User file ------------------------------------------------------------------
Match User file
        ForceCommand                internal-sftp
```

Quelltext 5.18: Verifikation „Dateiübertragungen“ - „Kommandozeilenzugriff bei Dateiübertragungen“ - Konfiguration für Benutzer `file`

#	Tätigkeit	Ergebnis
1	Auf Client `c1` mit folgendem Befehl als Benutzer `file` auf Server `s1` ein Login-Versuch unternommen: `ssh file@192.168.1.1`	Nach Eingabe der Key-Passphrase und des Benutzerkennworts wird folgender Text eingeblendet und die Verbindung getrennt: `This service allows sftp connections only.`

Tabelle 5.12.: Verifikation „Dateiübertragungen“ - „Kommandozeilenzugriff bei Dateiübertragungen“

5.4. Public-Key-Authentisierung

5.4.1. Nicht zugelassener Public-Key

Dieser Testfall prüft die Public-Key-Authentisierung mit und ohne zugelassenen Public-Key.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Die Datei `/home/cmd/.ssh/authorized_keys` ist leer (Ed25519-Public-Key von `user@c1` ist nicht hinterlegt)
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet

#	Tätigkeit	Ergebnis
1	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh -v cmd@192.168.1.1`	Die Verbindung wird nicht aufgebaut mit der Meldung, dass die Verbindung aufgrund des Public-Keys verweigert wurde siehe Quelltext 5.19
2	Auf Server `s1` wird unter `/home/cmd/.ssh/authorized_keys` der Public-Key von `user@c1` (den Inhalt von `/home/user/.ssh/id_ed25519.pub` auf Client `c1`) eingetragen (siehe Kapitel 4.2.1)	Public-Key von `user@c1` ist eingetragen
3	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh -v cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nun nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich

Tabelle 5.13.: Verifikation „Public-Key-Authentisierung“ - „Nicht zugelassener Public-Key“

```
c1$ ssh -v cmd@192.168.1.1
...
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Trying private key: /home/user/.ssh/id_rsa
debug1: Trying private key: /home/user/.ssh/id_ecdsa
debug1: Trying private key: /home/user/.ssh/id_ecdsa_sk
debug1: Offering public key: /home/user/.ssh/id_ed25519 ED25519 SHA256:
   SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk
debug1: Authentications that can continue: publickey
debug1: Trying private key: /home/user/.ssh/id_ed25519_sk
debug1: Trying private key: /home/user/.ssh/id_xmss
debug1: Trying private key: /home/user/.ssh/id_dsa
debug1: No more authentication methods to try.
cmd@192.168.1.1: Permission denied (publickey).
```

Quelltext 5.19: Verifikation „Public-Key-Authentisierung“ - „Nicht zugelassener Public-Key“ - Verbindungsversuch mit nicht eingetragenem Public-Key von `c1` zu `s1`

5.4.2. Zugelassener Public-Key mit unzulässigen Eigenschaften

Innerhalb dieses Testfalls wird geprüft, ob SSH-Verbindungen mit unzulässig ausgeprägten Schlüsseln aufgebaut werden können.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich folgende Schlüsseldateien:
    - DSA-Schlüssel: `id_dsa` und `id_dsa.pub`, erstellt mittels folgendem Befehl: `ssh-keygen -t dsa`
      Die Schlüssel werden mit einer Passphrase versehen und entsprechen einem nicht zugelassenen Schlüssel-Typen
    - 2048 Bit RSA-Schlüssel: `id_rsa` und `id_rsa.pub`, erstellt mittels folgendem Befehl: `ssh-keygen -t rsa -b 2048`
      Die Schlüssel werden mit einer Passphrase versehen und sind kleiner als die konfigurierte RSA-Schlüssel-Mindestgrösse von 3072 Bit
- Server `s1`
  - Die Public-Keys der zuvor generierten DSA- und RSA-Schlüssel von `user@c1` sind unter `/home/cmd/.ssh/authorized_keys` hinterlegt
    In dieser Datei befinden sich nur diese zwei Public-Keys
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet

#	Tätigkeit	Ergebnis
1	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` mit DSA-Schlüssel einloggen: `ssh -v -i id_dsa cmd@192.168.1.1`	Die Verbindung wird nicht aufgebaut mit der Meldung, dass die Verbindung aufgrund des Public-Keys verweigert wurde Es wird ausgegeben, dass der DSA-Schlüssel nicht verwendet wurde siehe Quelltext 5.20

2	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` mit RSA-Schlüssel einloggen:
	`ssh -v -i id_rsa cmd@192.168.1.1`
		Die Verbindung wird nicht aufgebaut mit der Meldung, dass die Verbindung aufgrund des Public-Keys verweigert wurde
		Es wird ausgegeben, dass der RSA-Schlüssel offeriert wurde
		siehe Quelltext 5.21
		
		Auf Server `s1` wird in `/var/log/authlog` folgendes aufgeführt:
		`refusing RSA key:`
		` Invalid key length [preauth]`

Tabelle 5.14.: Verifikation „Public-Key-Authentisierung“ - „Zugelassener Public-Key mit unzulässigen Eigenschaften“

```
c1$ ssh -v -i id_dsa cmd@192.168.1.1
...
debug1: identity file id_dsa type 1
debug1: identity file id_dsa-cert type -1
...
debug1: Skipping ssh-dss key id_dsa - corresponding algo not in
   PubkeyAcceptedAlgorithms
...
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: No more authentication methods to try.
cmd@192.168.1.1: Permission denied (publickey).
```

Quelltext 5.20: Verifikation „Public-Key-Authentisierung“ - „Zugelassener Public-Key mit unzulässigen Eigenschaften“ - Verbindungsversuch mit DSA-Schlüssel von `c1` zu `s1`

```
c1$ ssh -v -i id_rsa cmd@192.168.1.1
...
debug1: identity file id_rsa type 0
debug1: identity file id_rsa-cert type -1
...
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Offering public key: id_rsa RSA SHA256://
   gMKhN1cfVawNMx2qGfjKFNrZzX4t5LzPAxkwIJsoM explicit
debug1: Authentications that can continue: publickey
debug1: No more authentication methods to try.
cmd@192.168.1.1: Permission denied (publickey).
```

Quelltext 5.21: Verifikation „Public-Key-Authentisierung“ - „Zugelassener Public-Key mit unzulässigen Eigenschaften“ - Verbindungsversuch mit 2048-Bit-RSA-Schlüssel von `c1` zu `s1`

5.5. Jumphost

5.5.1. Konfiguration

Anhand dieses Testfalls wird das Verhalten der Konfiguration für Jumphosts betrachtet.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/jump/.ssh/authorized_keys` hinterlegt
  - Jumphost-Zugriff mittels Benutzer `jump` gemäss Kapitel 4.2.6, Quelltext 4.11 nicht implementiert
- Server `s2`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet

#	Tätigkeit	Ergebnis

1	Auf Client `c1` mit folgendem Befehl als Benutzer `jump` den Server `s1` als Jumphost genutzt und darüber als Benutzer `cmd` auf Server `s2` eingeloggt:
	`ssh -J jump@192.168.1.1 \`
	`cmd@192.168.2.1`
		Der Login als Benutzer `jump` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich, wird jedoch gleich wieder getrennt
		siehe Quelltext 5.22
		
		Auf Server `s1` wird in `/var/log/authlog` folgendes aufgeführt:
		`refused local port forward:`
		`originator 127.0.0.1 port 65535,`
		`target 192.168.2.1 port 22`

2	Auf Server `s1` in der SSH-Serverkonfiguration für Benutzer `jump` werden folgende Werte angefügt:
	`Match User jump`
	`DisableForwarding no`
	`AllowTcpForwarding yes`
	`PermitOpen 192.168.2.1:22`
	`MaxSessions 0`
		Die Konfiguration wird übernommen

3	Auf Client `c1` mit folgendem Befehl als Benutzer `jump` den Server `s1` als Jumphost genutzt und darüber als Benutzer `cmd` auf Server `s2` eingeloggt:
	`ssh -J jump@192.168.1.1 \`
	`cmd@192.168.2.1`
		Der Login ist nach Eingabe der Key-Passphrase und des Benutzerkennworts für jeweils den Jumphost (Benutzer `jump`) und den Zielserver (Benutzer `cmd`) erfolgreich
		siehe Quelltext 5.23

Tabelle 5.15.: Verifikation „Jumphost“ - „Konfiguration“

```
c1$ ssh -J jump@192.168.1.1 cmd@192.168.2.1
Enter passphrase for key '/home/user/.ssh/id_ed25519':
jump@192.168.1.1's password:
channel 0: open failed: administratively prohibited: open failed
stdio forwarding failed
kex_exchange_identification: Connection closed by remote host
Connection closed by UNKNOWN port 65535
```

Quelltext 5.22: Verifikation „Jumphost“ - „Konfiguration“ - Verbindungsversuch über Jumphost `s1` nach `s2`, initiiert von `c1` aus

```
c1$ ssh -J jump@192.168.1.1 cmd@192.168.2.1
Enter passphrase for key '/home/user/.ssh/id_ed25519':
jump@192.168.1.1's password:
Enter passphrase for key '/home/user/.ssh/id_ed25519':
cmd@192.168.2.1's password:
```

Quelltext 5.23: Verifikation „Jumphost“ - „Konfiguration“ - Verbindungsaufbau über Jumphost `s1` nach `s2`, initiiert von `c1` aus

5.5.2. Verbindung zu Server mit veralteter Version und nicht unterstützten Algorithmen

Dieser Test zeigt die Verbindung zu einem Server mit veraltetem oder nicht unterstütztem Stand über einen Jumphost auf.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/jump/.ssh/authorized_keys` hinterlegt
  - Jumphost-Zugriff mittels Benutzer `jump` gemäss Kapitel 4.2.6, Quelltext 4.11 implementiert mit der Anpassung, dass die Zeile mit `PermitOpen` nun wie folgt lautet: `PermitOpen 192.168.2.2:22`
- Debian 5 (Server `s3`) [76]
  mit Hostnamen „s3.lab.internal“ und IPv4-Adresse „192.168.2.2/24“
  Hierbei handelt es sich um eine neue VM in der Laborumgebung (siehe Kapitel 4.1), welche im selben Subnetz mit Server `s2` ist und mit Debian Version 5.0.8 vom 22.01.2011 [7] installiert wurde [77]
  Diese VM spielt die Rolle des alten Servers, welcher über eine veraltete OpenSSH-Version und -Konfiguration verfügt
  - Installation des OpenSSH-Servers mittels `apt-get install openssh-server` mit an der VM angehängter Debian-5-ISO-Datei [7]
  - Die OpenSSH-Server-Konfiguration wird auf der Standardeinstellung belassen und ist in Kapitel B.2 einsehbar
    Die anzutreffende OpenSSH-Version lautet `OpenSSH_5.1p1 Debian-5`, wobei `OpenSSH_5.1p1` am 22.07.2008 erschien [63]
  - Anlegen des Benutzers `cmd` inklusive Heimverzeichnis mittels `useradd -m cmd`
  - Definieren eines Passworts für den Benutzer `cmd`: `passwd cmd`
  - Erstellen der Datei `~/.ssh/authorized_keys` inkl. zugehöriger Ordner als Benutzer `cmd`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet
- Firewall
  - Firewall-Rule für die SSH-Verbindung von Server `s1` eingerichtet:
    Quelle: 192.168.1.1 (`s1`), Quellport beliebig
    Ziel: 192.168.2.2 (`s3`, Debian 5), Zielport: TCP-Port 22

[76] Aus Zeitgründen und vorhandener Erfahrung wurde eine veraltete Debian-Linux-Distribution anstelle einer alten OpenBSD-Version verwendet

[77] Das Debian-Betriebssystem wurde in Englisch, mit der Region „Switzerland“, dem Tastaturlayout „Swiss German“ und manueller IPv4-Adresskonfiguration installiert. Die Gateway- sowie DNS-Server-IP-Adresse entspricht der Firewall-IP-Adresse im selben Subnetz, also „192.168.2.254“. Die restliche Auswahl bei der Installation wurde entsprechend den vorgeschlagenen Standardeinstellungen übernommen, wobei sämtliche zu installierende Software abgewählt wurde

#	Tätigkeit	Ergebnis

1	Auf Client `c1` mit folgendem Befehl als Benutzer `jump` den Server `s1` als Jumphost genutzt und darüber als Benutzer `cmd` auf Server `s3` eingeloggt:
	`ssh -J jump@192.168.1.1 \`
	`cmd@192.168.2.2`
		Nach Eingabe der Key-Passphrase und des Benutzerkennworts für den Jumphost mit Benutzer `jump` wird die Verbindung wieder getrennt und folgendes aufgeführt:
		`Unable to negotiate with`
		`UNKNOWN port 65535:`
		`no matching host key type found.`
		`Their offer: ssh-rsa,ssh-dss`
		Laut Meldung gilt es einen der offerierten Host-Key-Typen zu wählen, wovon keiner standardmässig in der Option `HostKeyAlgorithms` des SSH-Clients hinterlegt ist. Dies wird im nächsten Schritt durchgeführt

2	Auf Client `c1` mit folgendem Befehl als Benutzer `jump` den Server `s1` als Jumphost genutzt und darüber als Benutzer `cmd` auf Server `s3` eingeloggt:
	`ssh -o "HostKeyAlgorithms +ssh-rsa" \`
	`-J jump@192.168.1.1 \`
	`cmd@192.168.2.2`
	Leider scheint es keinen Weg zu geben, die Option im `Match`-Block für `jump` auf `s1` zu konfigurieren, daher muss die Options-Anpassung vom Client aus vorgenommen werden
		Nach Eingabe der Key-Passphrase und des Benutzerkennworts für den Jumphost mit Benutzer `jump` sowie des Benutzerkennworts für den Debian-Server mit Benutzer `cmd` wurde die SSH-Verbindung erfolgreich aufgebaut
		siehe Quelltext 5.24

3	Auf Server `s3` in der OpenSSH-Server-Konfiguration unter `/etc/ssh/sshd_config` folgende Optionen definieren:
	`PasswordAuthentication no`
	`PubkeyAuthentication yes`
	Übernahme der Konfiguration mittels `/etc/init.d/ssh reload`
		Die Konfiguration wird übernommen
		Nun gilt es zu einem veralteten Server zu verbinden, welcher den Public-Key anstelle das Passwort prüft
		Die Option, mittels `AuthenticationMethods` mehrere Authentisierungsmethoden gleichzeitig vorauszusetzen, gibt es erst ab OpenSSH Version 6.2 vom 22.03.2013 [63]

4	Auf Client `c1` mit folgendem Befehl als Benutzer `jump` den Server `s1` als Jumphost genutzt und darüber als Benutzer `cmd` auf Server `s3` eingeloggt:
	`ssh -o "HostKeyAlgorithms +ssh-rsa" \`
	`-J jump@192.168.1.1 \`
	`cmd@192.168.2.2`
		Nach Eingabe der Key-Passphrase und des Benutzerkennworts für den Jumphost mit Benutzer `jump` wird die SSH-Verbindung abgebrochen mit der Meldung, dass die Verbindung aufgrund des Public-Keys verweigert wurde
		Erst bei Ausführen des SSH-Servers mit höchster Debugging-Ausgabe (siehe Tabelle 3.3 in Kapitel 3.4.3.1) und einem erneuten Versuch wird ausgegeben, dass der Ed25519-Schlüssel-Typ unbekannt ist
		siehe Quelltext 5.25

#	Tätigkeit	Ergebnis
5	Auf Client `c1` mit folgendem Befehl als Benutzer `user` einen 3072-Bit-RSA-Schlüssel unter `~/.ssh/id_rsa` und `~/.ssh/id_rsa.pub` generieren: `ssh-keygen -t rsa -b 3072` Die Schlüssel werden mit einer Passphrase versehen und sind kleiner als die konfigurierte RSA-Schlüssel-Mindestgrösse von 3072 Bit	RSA-Schlüssel ist generiert
6	RSA-Public-Key von `user@c1` auf `s3` unter `/home/cmd/.ssh/authorized_keys` hinterlegen und den Ed25519-Public-Key aus derselben Datei entfernen	Bei `cmd@s3` ist lediglich noch der RSA-Public-Key hinterlegt

7	Auf Client `c1` mit folgendem Befehl als Benutzer `jump` den Server `s1` als Jumphost genutzt und darüber als Benutzer `cmd` auf Server `s3` eingeloggt mit zusätzlicher Angabe des RSA-Keys:
	`ssh -v \`
	`-o "HostKeyAlgorithms +ssh-rsa" \`
	`-i id_rsa -J jump@192.168.1.1 \`
	`cmd@192.168.2.2`
		Nach Eingabe der Key-Passphrase und des Benutzerkennworts für den Jumphost mit Benutzer `jump` wird die SSH-Verbindung mit folgender Meldung abgebrochen:
		`send_pubkey_test:`
		`no mutual signature algorithm`
		Auf dem Client werden unterstützte Signatur Algorithmen mit `ssh -Q sig` abgefragt und mit den derzeitigen Werten anhand
		`ssh -G -o "HostKeyAlgorithms +ssh-rsa" \`
		`-i id_rsa -J jump@192.168.1.1 \`
		`cmd@192.168.2.2 | grep pubkey` geprüft
		→ `ssh-rsa` fehlt

8	Auf Client `c1` mit folgendem Befehl als Benutzer `jump` den Server `s1` als Jumphost genutzt und darüber als Benutzer `cmd` auf Server `s3` eingeloggt mit zusätzlicher Angabe des RSA-Keys:
	`ssh -v \`
	`-o "HostKeyAlgorithms +ssh-rsa" \`
	`-o "PubkeyAcceptedAlgorithms \`
	`+ssh-rsa" -i id_rsa \`
	`-J jump@192.168.1.1 \`
	`cmd@192.168.2.2`
		Nach Eingabe der Ed25519-Key-Passphrase und des Benutzerkennworts für den Jumphost mit Benutzer `jump` und der RSA-Key-Passphrase für `s3` mit Benutzer `cmd` wird die SSH-Verbindung aufgebaut
		Aus der Ausgabe wird ersichtlich, dass der Jumphost `s1` den RSA-Public-Key nicht akzeptiert, da dieser nicht bei `jump@s1` hinterlegt ist, jedoch daraufhin den hinterlegten Ed25519-Public-Key offeriert erhält und diesen akzeptiert
		Für die Verbindung zum Zielserver wird wieder der RSA-Public-Key offeriert und diesmal für `cmd@s3` akzeptiert
		siehe Quelltext 5.26

Tabelle 5.16.: Verifikation „Jumphost“ - „Verbindung zu Server mit veralteter Version und nicht unterstützten Algorithmen“

```
c1$ ssh -o "HostKeyAlgorithms +ssh-rsa" -J jump@192.168.1.1 cmd@192.168.2.2
Enter passphrase for key '/home/user/.ssh/id_ed25519':
jump@192.168.1.1's password:
The authenticity of host '192.168.2.2 (<no hostip for proxy command>)' can't
   be established.
RSA key fingerprint is SHA256:tZcia9cPFRLwKbm5NVK3/dJ5YyJHO7XCa8tVwBLfqCo.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.2.2' (RSA) to the list of known hosts.
cmd@192.168.2.2's password:
Linux debian 2.6.26-2-amd64 #1 SMP Thu Nov 25 04:30:55 UTC 2010 x86_64
...
cmd@s3:~$
```

Quelltext 5.24: Verifikation „Jumphost“ - „Verbindung zu Server mit veralteter Version und nicht unterstützten Algorithmen“ - Verbindungsaufbau über Jumphost `s1` nach `s3` mit zusätzlichem, alten Host-Key-Typen, initiiert von `c1` aus

```
...
debug1: userauth-request for user cmd service ssh-connection method publickey
debug1: attempt 1 failures 0
debug2: input_userauth_request: try method publickey
debug2: key_type_from_name: unknown key type 'ssh-ed25519'
userauth_pubkey: unsupported public key algorithm: ssh-ed25519
debug2: userauth_pubkey: authenticated 0 pkalg ssh-ed25519
...
```

Quelltext 5.25: Verifikation „Jumphost“ - „Verbindung zu Server mit veralteter Version und nicht unterstützten Algorithmen“ - Verbindungsaufbau über Jumphost `s1` nach `s3`, initiiert von `c1` aus - Debugging-Ausgabe auf `s3`

```
c1$ ssh -v -o "HostKeyAlgorithms +ssh-rsa" -o "PubkeyAcceptedAlgorithms +ssh-
   rsa" -i id_rsa -J jump@192.168.1.1 cmd@192.168.2.2
...
debug1: Connecting to 192.168.1.1 [192.168.1.1] port 22.
debug1: identity file id_rsa type 0
debug1: identity file id_rsa-cert type -1
...
debug1: Authenticating to 192.168.1.1:22 as 'jump'
...
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Offering public key: /home/user/.ssh/id_rsa RSA SHA256:
   a6QoLg62HCnDwotM+OKzhQuKZ+uba2bppEV0lYujWPc
debug1: Authentications that can continue: publickey
debug1: Trying private key: /home/user/.ssh/id_ecdsa
debug1: Trying private key: /home/user/.ssh/id_ecdsa_sk
debug1: Offering public key: /home/user/.ssh/id_ed25519 ED25519 SHA256:
   SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk
debug1: Server accepts key: /home/user/.ssh/id_ed25519 ED25519 SHA256:
   SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk
Enter passphrase for key '/home/user/.ssh/id_ed25519':
Authenticated using "publickey" with partial success.
debug1: Authentications that can continue: password
debug1: Next authentication method: password
jump@192.168.1.1's password:
Authenticated to 192.168.1.1 ([192.168.1.1]:22) using "password".
debug1: channel_connect_stdio_fwd: 192.168.2.2:22
...
debug1: compat_banner: match: OpenSSH_5.1p1 \gls{gloss:debian}-5 pat OpenSSH_5
   * compat 0x0c000002
debug1: Authenticating to 192.168.2.2:22 as 'cmd'
```

```
...
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Offering public key: id_rsa RSA SHA256:a6QoLg62HCnDwotM+OKzhQuKZ+
   uba2bppEV0lYujWPc explicit
debug1: Server accepts key: id_rsa RSA SHA256:a6QoLg62HCnDwotM+OKzhQuKZ+
   uba2bppEV0lYujWPc explicit
Enter passphrase for key 'id_rsa':
Authenticated to 192.168.2.2 (via proxy) using "publickey".
...
cmd@s3:~$
```

Quelltext 5.26: Verifikation „Jumphost“ - „Verbindung zu Server mit veralteter Version und nicht unterstützten Algorithmen“ - Verbindungsaufbau über Jumphost `s1` nach `s3`, initiiert von `c1` aus - RSA-Public-Key nur bei `cmd@s3` und nicht bei `jump@s1` hinterlegt

Somit ist hiermit der Zugang zu einem älteren OpenSSH-Server am Beispiel einer Debian-5-Installation geprüft.

Anstelle des Befehls aus dem letzten Schritt können die Optionen für den Zielhost in der SSH-Client-Konfiguration unter `~/.ssh/config` wie folgt hinterlegt werden:

```
Host 192.168.2.2
    ProxyJump jump@192.168.1.1
    HostKeyAlgorithms +ssh-rsa
    PubkeyAcceptedAlgorithms +ssh-rsa
    IdentityFile id_rsa
    User cmd
```

Quelltext 5.27: Verifikation „Jumphost“ - „Verbindung zu Server mit veralteter Version und nicht unterstützten Algorithmen“ - Beispiel einer SSH-Client-Konfiguration unter `~/.ssh/config` zur Vereinfachung der Befehlseingabe

Nun kann anstelle des ausführlichen Befehls die SSH-Verbindung über den Jumphost `s1` mit einer vom Zielhost akzeptierten Algorithmen-Wahl wie folgt durchgeführt werden: `ssh 192.168.2.2` Daraufhin werden sämtliche Parameter aus der Client-Konfiguration gelesen und angewendet [78].

Gilt es zusätzlich für unterschiedliche Hosts unterschiedliche Schlüssel zu verwenden, welche nicht im lokalen `.ssh`-Verzeichnis liegen, werden diese nicht standardmässig durchprobiert [79]. Stattdessen gilt es dann den entsprechenden Schlüssel mittels `IdentityFile`-Option in der zuvor erwähnten Client-Konfiguration zu hinterlegen, in welcher z.B. pro Host ein spezifischer Schlüssel angegeben werden kann [80].

[78] Mehr zur SSH-Client-Konfiguration ist in der Manpage von `ssh_config` [57] vorzufinden

[79] Bei diesem Testfall lag im letzten Schritt der Schlüssel bereits im `.ssh`-Verzeichnis, weshalb er erkannt und verwendet wurde. Zudem muss der Dateiname des Schlüssels in `.ssh` einem spezifischen Namen entsprechen, siehe Option `IdentityFile` in der `ssh_config`-Manpage [57]

[80] Wird im Testfall in Kapitel 5.9.2 demonstriert

5.6. Authentisierungs-Agent

5.6.1. Konfiguration

Anhand dieses Testfalls wird das Verhalten eines Authentisierungs-Agenten betrachtet.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet

#	Tätigkeit	Ergebnis
1	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich
2	Die Sitzung wird durch Eingabe von `exit` beendet	Sitzung mit `s1` wird beendet

3	Auf Client `c1` den Authentisierungs-Agenten starten und den Ed25519-Schlüssel gemäss Kapitel 4.2.7 hinzufügen:
	`eval $(ssh-agent -s)`
	`ssh-add ~/.ssh/id_ed25519`
		`ssh-agent` wurde gestartet und der Ed25519-Schlüssel nach Eingabe von dessen Passphrase hinzugefügt
		siehe Quelltext 5.28

4	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nach Eingabe des Benutzerkennworts erfolgreich, nach der Key-Passphrase wird nicht gefragt

Tabelle 5.17.: Verifikation „Authentisierungs-Agent“ - „Konfiguration“

```
1 c1$ eval $(ssh-agent -s)
2 Agent pid 25432
3 c1$ ssh-add ~/.ssh/id_ed25519
4 Enter passphrase for /home/user/.ssh/id_ed25519:
5 Identity added: /home/user/.ssh/id_ed25519 (user@c1.lab.internal)
```

Quelltext 5.28: Verifikation „Authentisierungs-Agent“ - „Konfiguration“ - Starten des Authentisierungs-Agenten und Hinzufügen eines Ed25519-Schlüssels

5.6.2. Agent-Forwarding

Dieser Test veranschaulicht die Agent-Forwarding-Funktionalität.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
  - Der Authentisierungs-Agent ist gemäss vorherigem Test in Kapitel 5.6.1 gestartet und hat den zuvor erwähnten Ed25519-Schlüssel geladen
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` und unter `/home/agent/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet
  - Jumphost-Zugriff mittels Benutzer `jump` gemäss Kapitel 4.2.6, Quelltext 4.11 implementiert
  - Agent-Forwarding für Benutzer `agent` gemäss Kapitel 4.2.7.1, Quelltext 4.17 implementiert
    Für den Benutzer `agent` ist in der Server-Konfiguration dasselbe wie für den Benutzer `cmd` hinterlegt, mit dem zusätzlichen Parameter `AllowAgentForwarding yes` als Unterschied
- Server `s2`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet

#	Tätigkeit	Ergebnis

1	Auf Client `c1` mit folgendem Befehl als Benutzer `jump` den Server `s1` als Jumphost genutzt und darüber als Benutzer `cmd` auf Server `s2` eingeloggt:
	`ssh -J jump@192.168.1.1 \`
	`cmd@192.168.2.1`
		Der Login ist nach Eingabe der Benutzerkennwörter für jeweils den Jumphost (Benutzer `jump`) und den Zielserver (Benutzer `cmd`) erfolgreich, nach der Key-Passphrase wird nicht gefragt

2	Die Sitzung wird durch Eingabe von `exit` beendet	Sitzung mit `s1` wird beendet
3	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` mit aktiviertem Agent-Forwarding auf dem Server `s1` einloggen: `ssh -A cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nach Eingabe des Benutzerkennworts erfolgreich

4	In der SSH-Sitzung mit `cmd@192.168.1.1` mit folgendem Befehl die im Agent geladenen Keys prüfen:
	`ssh-add -L`
		Es wird kein geladener Schlüssel angezeigt, dafür die Meldung
		`Could not open a connection`
		`to your authentication agent.`

5	In der SSH-Sitzung mit `cmd@192.168.1.1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s2` einloggen: `ssh -v cmd@192.168.2.1`	Der Login ist nicht aufgrund des fehlenden Schlüssels nicht erfolgreich siehe Quelltext 5.29

#	Tätigkeit	Ergebnis
6	Die Sitzung wird durch Eingabe von `exit` beendet	Sitzung mit `s1` wird beendet
7	Auf Client `c1` mit folgendem Befehl als Benutzer `agent` mit aktiviertem Agent-Forwarding auf dem Server `s1` einloggen: `ssh -A agent@192.168.1.1`	Der Login als Benutzer `agent` ist nach Eingabe des Benutzerkennworts erfolgreich
8	In der SSH-Sitzung mit `agent@192.168.1.1` mit folgendem Befehl die im Agent geladenen Keys prüfen: `ssh-add -L`	Der Ed25519-Public-Key wird angezeigt
9	In der SSH-Sitzung mit `agent@192.168.1.1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s2` einloggen: `ssh -v cmd@192.168.2.1`	Der Login als Benutzer `cmd` ist nach Eingabe des Benutzerkennworts erfolgreich siehe Quelltext 5.30

Tabelle 5.18.: Verifikation „Authentisierungs-Agent“ - „Agent-Forwarding“

```
s1$ ssh -v cmd@192.168.2.1
...
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Trying private key: /home/cmd/.ssh/id_rsa
debug1: Trying private key: /home/cmd/.ssh/id_ecdsa
debug1: Trying private key: /home/cmd/.ssh/id_ecdsa_sk
debug1: Trying private key: /home/cmd/.ssh/id_ed25519
debug1: Trying private key: /home/cmd/.ssh/id_ed25519_sk
debug1: Trying private key: /home/cmd/.ssh/id_xmss
debug1: Trying private key: /home/cmd/.ssh/id_dsa
debug1: No more authentication methods to try.
cmd@192.168.2.1: Permission denied (publickey).
```

Quelltext 5.29: Verifikation „Authentisierungs-Agent“ - „Agent-Forwarding“ - Verbindungsversuch ohne aktives Agent-Forwarding (kein Schlüssel geladen)

```
s1$ ssh -v cmd@192.168.2.1
...
debug1: Offering public key: user@c1.lab.internal ED25519 SHA256:SQGlWRYdvL/
    NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk agent
debug1: Server accepts key: user@c1.lab.internal ED25519 SHA256:SQGlWRYdvL/
    NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk agent
Authenticated using "publickey" with partial success.
debug1: Authentications that can continue: password
debug1: Next authentication method: password
cmd@192.168.2.1's password:
Authenticated to 192.168.2.1 ([192.168.2.1]:22) using "password".
...
s2$
```

Quelltext 5.30: Verifikation „Authentisierungs-Agent“ - „Agent-Forwarding“ - Verbindungsversuch mit aktivem Agent-Forwarding und geladenem Schlüssel

5.6.3. Eingeschränkte Key-Nutzung bei Agent-Forwarding

Anhand dieses Testfalls wird die Einschränkung der Nutzung von in einem Agent geladenen Key betrachtet.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
  - Der Authentisierungs-Agent ist gemäss Test in Kapitel 5.6.1 gestartet und hat keinen Schlüssel geladen
    Kontrolle mittels `ssh-add -L`, wessen Ausgabe wie folgt lauten soll: `The agent has no identities.`
  - Die Datei `~/.ssh/known_hosts` soll die Host-Public-Keys von Server `s1` und `s2` beinhalten (siehe Kapitel 4.2.7.2), wie in folgendem Beispiel:
    ```
    192.168.1.1 ssh-ed25519 AAAAC3NzaC1lZD...
    192.168.1.1 ssh-rsa AAAAB3NzaC1yc2...
    192.168.1.1 ecdsa-sha2-nistp256 AAAAE2VjZHNhLX...
    192.168.2.1 ssh-ed25519 AAAAC3NzaC1lZD...
    192.168.2.1 ssh-rsa AAAAB3NzaC1yc2...
    192.168.2.1 ecdsa-sha2-nistp256 AAAAE2VjZHNhLX...
    ```
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` und unter `/home/agent/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet
  - Agent-Forwarding für Benutzer `agent` gemäss Kapitel 4.2.7.1, Quelltext 4.17 implementiert
    Für den Benutzer `agent` ist in der Server-Konfiguration dasselbe wie für den Benutzer `cmd` hinterlegt, mit dem zusätzlichen Parameter `AllowAgentForwarding yes` als Unterschied
- Server `s2`
  - Ed25519-Public-Key von `user@c1` unter `/home/cmd/.ssh/authorized_keys` und unter `/home/agent/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `agent` und `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet
    Ein Agent-Forwarding muss hier nicht aktiviert werden, da dies der letzte Host in der Kette ist

#	Tätigkeit	Ergebnis

1	Auf Client `c1` mit folgendem Befehl gemäss Kapitel 4.2.7.2 den Schlüssel in den Authentisierungs-Agenten laden, sodass er nur für die Verbindung zu `agent@192.168.1.1` und der Sitzung von `agent@192.168.1.1` nach `agent@192.168.2.1` erlaubt ist:
	`ssh-add -h 'agent@192.168.1.1' \`
	`-h '192.168.1.1>agent@192.168.2.1' \`
	`~/.ssh/id_ed25519`
		Der Schlüssel wurde nach Eingabe von dessen Passphrase mit entsprechenden Einschränkungen in den Authentisierungs-Agenten geladen

#	Tätigkeit	Ergebnis
2	Auf Client `c1` mit folgendem Befehl die im Agent geladenen Keys prüfen: `ssh-add -L`	Der Ed25519-Public-Key wird angezeigt Die definierten Einschränkungen werden nicht aufgeführt
3	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` mit aktiviertem Agent-Forwarding auf dem Server `s1` einloggen: `ssh -A cmd@192.168.1.1`	Der Login schlägt mit der Meldung fehl, dass der Agent den Zugriff verweigert hat Dasselbe Ergebnis wird ohne Parameter `-A` erzielt siehe Quelltext 5.31
4	Auf Client `c1` mit folgendem Befehl als Benutzer `agent` mit aktiviertem Agent-Forwarding auf dem Server `s1` einloggen: `ssh -A agent@192.168.1.1`	Der Login als Benutzer `agent` ist nach Eingabe des Benutzerkennworts erfolgreich
5	In der SSH-Sitzung mit `agent@192.168.1.1` mit folgendem Befehl die im Agent geladenen Keys prüfen: `ssh-add -L`	Der Ed25519-Public-Key wird angezeigt Die definierten Einschränkungen werden nicht aufgeführt
6	In der SSH-Sitzung mit `agent@192.168.1.1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s2` einloggen: `ssh -v cmd@192.168.2.1`	Der Login schlägt mit der Meldung fehl, dass der Agent den Zugriff verweigert hat siehe Quelltext 5.32
7	In der SSH-Sitzung mit `agent@192.168.1.1` mit folgendem Befehl als Benutzer `agent` auf dem Server `s2` einloggen: `ssh -v agent@192.168.2.1`	Der Login als Benutzer `agent` ist nach Eingabe des Benutzerkennworts erfolgreich

Tabelle 5.19.: Verifikation „Authentisierungs-Agent“ - „Eingeschränkte Key-Nutzung bei Agent-Forwarding“

```
1 c1$ ssh -A cmd@192.168.1.1
2 sign_and_send_pubkey: signing failed for ED25519 "/home/user/.ssh/id_ed25519"
     from agent: agent refused operation
3 cmd@192.168.1.1: Permission denied (publickey).
```

Quelltext 5.31: Verifikation „Authentisierungs-Agent“ - „Eingeschränkte Key-Nutzung bei Agent-Forwarding“ - Verbindungsversuch mit Agent-Forwarding zu nicht erlaubter Verbindung von Client `c1`

```
1 s1$ ssh cmd@192.168.2.1
2 sign_and_send_pubkey: signing failed for ED25519 "user@c1.lab.internal" from
     agent: agent refused operation
3 cmd@192.168.2.1: Permission denied (publickey).
```

Quelltext 5.32: Verifikation „Authentisierungs-Agent“ - „Eingeschränkte Key-Nutzung bei Agent-Forwarding“ - Verbindungsversuch mit Agent-Forwarding zu nicht erlaubter Verbindung von Server `s1` mit Agent-Forwarding

5.7. Zertifikate

5.7.1. Hinterlegung der CA

Dieser Testfall prüft die Zertifikatsauthentisierung mit Zertifikaten einer selbst hinterlegten CA.

Vorbedingungen:

- CA
  - CA gemäss Kapitel 4.2.8.1 erstellt
- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Datei `/home/cmd/.ssh/authorized_principals` gemäss Kapitel 4.2.8.2, Quelltext 4.23 hinterlegt
    Enthält Zeilen `principala` und `principalb`
  - Datei `/home/cmd/.ssh/authorized_principals` ist die einzige Datei im Ordner `/home/cmd/.ssh`
    Restliche Dateien löschen
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.7 eingerichtet
  - Zertifikatsauthentisierung
    - Zertifikatsauthentisierung gemäss Kapitel 4.2.8.2, Quelltext 4.22 eingerichtet
    - Zertifikatsauthentisierung gemäss Kapitel 4.2.8.5, Quelltext 4.28 forciert
      Die Option `AuthorizedKeysFile` entspricht dem Wert `none`
    - CA-Public-Key ist gemäss Kapitel 4.2.8.2, Quelltext 4.21 unter `/etc/ssh/ca.pub` hinterlegt

#	Tätigkeit	Ergebnis

1	Auf Client `c1` mit folgendem Befehl ein selbst signiertes Zertifikat des Public-Keys `id_ed25519.pub` mit Principal `principala` und dem eigenen Ed25519-Schlüssel erstellen:
	`ssh-keygen -s ~/.ssh/id_ed25519 \`
	`-n principala \`
	`-I user_c1_20230915_selfsigned \`
	`~/.ssh/id_ed25519.pub`
		Zertifikat ist unter `/home/user/.ssh/id_ed25519-cert.pub` erstellt

#	Tätigkeit	Ergebnis
2	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh -v cmd@192.168.1.1`	Die Verbindung wird nicht aufgebaut mit der Meldung, dass die Verbindung aufgrund des Public-Keys verweigert wurde Das selbst signierte Zertifikat wird offeriert, aber nicht akzeptiert Ohne ein hinterlegtes Zertifikat wird dasselbe Ergebnis erzielt, jedoch wird kein Zertifikat bei der Authentisierung angeboten siehe Quelltext 5.33
3	Auf Client `c1` die Datei `/home/user/.ssh/id_ed25519-cert.pub` löschen	Datei ist gelöscht

4	Auf der CA den Public-Key `id_ed25519.pub` mit Principal `principala` signieren und daraus resultierendes Zertifikat auf Client `c1` unter `/home/user/.ssh/id_ed25519-cert.pub` abspeichern (siehe Kapitel 4.2.8.3, Quelltext 4.24):
	`ssh-keygen -s ~/ca/ca \`
	`-n principala \`
	`-I user_c1_20230915_principala \`
	`id_ed25519.pub`
		Zertifikat von CA ohne Principal ist nach Eingabe der CA-Key-Passphrase generiert und auf `c1` abgespeichert

5	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh -v cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich

Tabelle 5.20.: Verifikation „Zertifikate“ - „Hinterlegung der CA“

```
c1$ ssh -v cmd@192.168.1.1
...
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Trying private key: /home/user/.ssh/id_rsa
debug1: Trying private key: /home/user/.ssh/id_ecdsa
debug1: Trying private key: /home/user/.ssh/id_ecdsa_sk
debug1: Offering public key: /home/user/.ssh/id_ed25519 ED25519 SHA256:
    SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk
debug1: Authentications that can continue: publickey
debug1: Offering public key: /home/user/.ssh/id_ed25519 ED25519-CERT SHA256:
    SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk
debug1: Authentications that can continue: publickey
debug1: Trying private key: /home/user/.ssh/id_ed25519_sk
debug1: Trying private key: /home/user/.ssh/id_xmss
debug1: Trying private key: /home/user/.ssh/id_dsa
debug1: No more authentication methods to try.
cmd@192.168.1.1: Permission denied (publickey).
```

Quelltext 5.33: Verifikation „Zertifikate“ - „Hinterlegung der CA“ - Verbindungsversuch mit ungültigem Zertifikat von `c1` zu `s1`

5.7.2. Principals

Anhand dieses Testfalls wird die Zertifikatsauthentisierung im Zusammenhang mit Principals geprüft, sodass ein Benutzer ein Zertifikat nur zur Authentisierung verwenden kann, wenn ein passender Principal hinterlegt ist.

Vorbedingungen:

- CA
  - CA gemäss Kapitel 4.2.8.1 erstellt
- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
- Server `s1`
  - Datei `/home/cmd/.ssh/authorized_principals` gemäss Kapitel 4.2.8.2, Quelltext 4.23 hinterlegt
    Enthält Zeilen `principala` und `principalb`
  - Datei `/home/cmd/.ssh/authorized_principals` ist die einzige Datei im Ordner `/home/cmd/.ssh`
    Restliche Dateien löschen
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.7 eingerichtet
  - Zertifikatsauthentisierung
    - Zertifikatsauthentisierung gemäss Kapitel 4.2.8.2, Quelltext 4.22 eingerichtet
    - Zertifikatsauthentisierung gemäss Kapitel 4.2.8.5, Quelltext 4.28 forciert
      Die Option `AuthorizedKeysFile` entspricht dem Wert `none`
    - CA-Public-Key ist gemäss Kapitel 4.2.8.2, Quelltext 4.21 unter `/etc/ssh/ca.pub` hinterlegt

#	Tätigkeit	Ergebnis
1	Auf der CA den Public-Key `id_ed25519.pub` ohne Principal signieren und daraus resultierendes Zertifikat auf Client `c1` unter `/home/user/.ssh/id_ed25519-cert.pub` abspeichern (siehe Kapitel 4.2.8.3, Quelltext 4.24): `ssh-keygen -s ~/ca/ca -I user_c1_20230915_noprincipal id_ed25519.pub`	Zertifikat von CA ohne Principal ist nach Eingabe der CA-Key-Passphrase generiert und auf `c1` abgespeichert
2	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh -v cmd@192.168.1.1`	Die Verbindung wird nicht aufgebaut mit der Meldung, dass die Verbindung aufgrund des Public-Keys verweigert wurde Das Zertifikat von der CA wird offeriert, aber nicht akzeptiert In der Datei `/var/log/authlog` auf dem Server `s1` ist folgender Eintrag ersichtlich: `error: Certificate does not contain an authorized principal`

#	Tätigkeit	Ergebnis
3	Auf Client `c1` die Datei `/home/user/.ssh/id_ed25519-cert.pub` löschen	Datei ist gelöscht

4	Auf der CA den Public-Key `id_ed25519.pub` mit nicht hinterlegtem Principal signieren und daraus resultierendes Zertifikat auf Client `c1` unter `/home/user/.ssh/id_ed25519-cert.pub` abspeichern (siehe Kapitel 4.2.8.3, Quelltext 4.24):
	`ssh-keygen -s ~/ca/ca \`
	`-n invalidprinc \`
	`-I user_c1_20230915_invalidprinc \`
	`id_ed25519.pub`
		Zertifikat von CA mit nicht hinterlegtem Principal ist nach Eingabe der CA-Key-Passphrase generiert und auf `c1` abgespeichert

5	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh -v cmd@192.168.1.1`	Die Verbindung wird nicht aufgebaut mit der Meldung, dass die Verbindung aufgrund des Public-Keys verweigert wurde Das Zertifikat von der CA wird offeriert, aber nicht akzeptiert In der Datei `/var/log/authlog` auf dem Server `s1` ist folgender Eintrag ersichtlich: `error: Certificate does not contain an authorized principal`
6	Auf Client `c1` die Datei `/home/user/.ssh/id_ed25519-cert.pub` löschen	Datei ist gelöscht

7	Auf der CA den Public-Key `id_ed25519.pub` mit nicht hinterlegtem und hinterlegtem Principal signieren und daraus resultierendes Zertifikat auf Client `c1` unter `/home/user/.ssh/id_ed25519-cert.pub` abspeichern (siehe Kapitel 4.2.8.3, Quelltext 4.24):
	`ssh-keygen -s ~/ca/ca \`
	`-n principala,invalidprinc \`
	`-I user_c1_20230915_a_and_invalidpri`
	`id_ed25519.pub`
		Zertifikat von CA mit nicht hinterlegtem und hinterlegtem Principal ist nach Eingabe der CA-Key-Passphrase generiert und auf `c1` abgespeichert

8	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh -v cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich Dasselbe Ergebnis erhält man beim Verwenden eines Zertifikats, bei welchem nur ein hinterlegter Principal verwendet wird Es muss lediglich einer der Principals im Zertifikat passen

Tabelle 5.21.: Verifikation „Zertifikate“ - „Principals“

5.7.3. Host-Zertifikate

Dieser Test zeigt das Verhalten des SSH-Clients auf, wenn von dem SSH-Server Zertifikate offeriert werden, welche von einer vertrauten CA abstammen.

Vorbedingungen:

- CA
  - CA gemäss Kapitel 4.2.8.1 erstellt
- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet
    - die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`
    - ein zugehöriges Zertifikat `id_ed25519-cert.pub` mit auf dem Server hinterlegtem Principal (z.B. `principalb`)
- Server `s1`
  - Datei `/home/cmd/.ssh/authorized_principals` gemäss Kapitel 4.2.8.2, Quelltext 4.23 hinterlegt
    Enthält Zeilen `principala` und `principalb`
  - Datei `/home/cmd/.ssh/authorized_principals` ist die einzige Datei im Ordner `/home/cmd/.ssh`
    Restliche Dateien löschen
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.7 eingerichtet
  - Unter `/etc/ssh` befinden sich keine Zertifikate
  - Zertifikatsauthentisierung
    - Zertifikatsauthentisierung gemäss Kapitel 4.2.8.2, Quelltext 4.22 eingerichtet
    - Zertifikatsauthentisierung gemäss Kapitel 4.2.8.5, Quelltext 4.28 forciert
      Die Option `AuthorizedKeysFile` entspricht dem Wert `none`
    - CA-Public-Key ist gemäss Kapitel 4.2.8.2, Quelltext 4.21 unter `/etc/ssh/ca.pub` hinterlegt

#	Tätigkeit	Ergebnis
1	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich Der Host muss zuvor anhand des Fingerprints bestätigt werden
2	Die Sitzung wird durch Eingabe von `exit` beendet	Sitzung mit `s1` wird beendet

3	Auf der CA die Public-Keys des Servers `s1` `ssh_host_ecdsa_key.pub`, `ssh_host_ed25519_key.pub` und `ssh_host_rsa_key.pub` signieren und daraus resultierende Zertifikate auf `s1` unter `/etc/ssh/` abspeichern (siehe Kapitel 4.2.8.6, Quelltext 4.29):
	`ssh-keygen -s ~/ca/ca \`
	`-I host_s1 -h ssh_host_*.pub`
		Zertifikate von CA werden nach Eingabe der CA-Key-Passphrase generiert und sind auf `s1` unter `/etc/ssh/` abgespeichert

4	Auf Server `s1` in der OpenSSH-Server-Konfiguration unter `/etc/ssh/sshd_config` die erstellten Zertifikate der CA gemäss Kapitel 4.2.8.6, Quelltext 4.31 mittels Option `HostCertificate` hinterlegen	Die Konfiguration wird übernommen
5	Auf Client `c1` die Datei `~/.ssh/known_hosts` so anpassen, dass diese lediglich den CA-Public-Key gemäss Kapitel 4.2.8.6, Quelltext 4.32 hinterlegt hat	Datei ist angepasst
6	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh -v cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich Der Host muss anhand des Fingerprints nicht auf dem Client bestätigt werden siehe Quelltext 5.34
7	Die Sitzung wird durch Eingabe von `exit` beendet	Sitzung mit `s1` wird beendet
8	Auf Client `c1` den Inhalt der Datei `~/.ssh/known_hosts` prüfen	Es ist immer noch nur der CA-Public-Key hinterlegt, der Public-Key des Servers wurde nicht hinterlegt

Tabelle 5.22.: Verifikation „Zertifikate“ - „Host-Zertifikate“

```
c1$ ssh -v cmd@192.168.1.1
...
debug1: Server host certificate: ssh-ed25519-cert-v01@openssh.com SHA256:
    ctWpch78E8V8qZEv74v72HQaHVTauCgB0xF5xF1OvRU, serial 0 ID "host_s1" CA ssh-
    ed25519 SHA256:GwSQXlRbLXCg51XIVqJatsqKZL8JZxr91tBNRU5DS3Q valid forever
..
debug1: Host '192.168.1.1' is known and matches the ED25519-CERT host
    certificate.
debug1: Found CA key in /home/user/.ssh/known_hosts:1
..
debug1: Offering public key: /home/user/.ssh/id_ed25519 ED25519 SHA256:
    SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk
debug1: Authentications that can continue: publickey
debug1: Offering public key: /home/user/.ssh/id_ed25519 ED25519-CERT SHA256:
    SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk
debug1: Server accepts key: /home/user/.ssh/id_ed25519 ED25519-CERT SHA256:
    SQGlWRYdvL/NjJ3pPJPNe0OotzR7RSslQ5pe/QJ+CEk
Enter passphrase for key '/home/user/.ssh/id_ed25519':
Authenticated using "publickey" with partial success.
debug1: Authentications that can continue: password
debug1: Next authentication method: password
cmd@192.168.1.1's password:
Authenticated to 192.168.1.1 ([192.168.1.1]:22) using "password".
...
s1$
```

Quelltext 5.34: Verifikation „Zertifikate“ - „Host-Zertifikate“ - Verbindungsaufbau mit hinterlegtem Host-Zertifikat von `c1` zu `s1`

5.8. SSHFP-DNS-Records

5.8.1. Fingerprint in SSHFP-DNS-Record

Dieser Testfall zeigt das Verifizieren von SSHFP-DNS-Records. Da in Quelltext 4.37 in Kapitel 4.2.9.2 bereits gezeigt wird, dass beim Aufbau einer SSH-Verbindung mit einer IP-Adresse als Ziel keine SSHFP-DNS-Verifizierung durchgeführt wird, werden die Hosts in diesem Testfall ausschliesslich mit ihrem FQDN aufgerufen.

Vorbedingungen:

- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` beinhaltet lediglich die in Kapitel 4.2.1 erstellten Ed25519-Schlüsseldateien `id_ed25519` und `id_ed25519.pub`, keine `known_hosts`-Dateien
  - DNS-Server gemäss Kapitel 4.2.9.1 hinterlegt, kann Einträge der Zonen `lab.internal` und `168.192.in-addr.arpa` auflösen
- Server `s1`
  - Ed25519-Public-Key von `user@c1` unter `/home/jump/.ssh/authorized_keys` hinterlegt
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet
  - Einsatz eines DNS-Servers mit Konfiguration, Zonen und SSHFP-Records gemäss Kapitel 4.2.9.1

#	Tätigkeit	Ergebnis

1	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` mit aktiver DNS-SSHFP-Überprüfung einloggen:
	`ssh -o "VerifyHostKeyDNS yes" \`
	`cmd@s1.lab.internal`
		Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich
		Der Host muss zuvor anhand des Fingerprints bestätigt werden, wobei zusätzlich gemeldet wird, dass der passende Fingerprint mittels DNS gefunden wurde
		siehe Quelltext 5.35

2	Die Sitzung wird durch Eingabe von `exit` beendet	Sitzung mit `s1` wird beendet
3	Auf Server `s1` die bestehende DNS-Zone für `lab.internal` mittels folgendem Befehl kopieren: `cp /var/nsd/zones/master/\` `lab.internal.zone /tmp/`	Die Zonen-Datei wurde kopiert
4	Auf Server `s1` die bestehende DNS-Zone für `lab.internal` unter `/var/nsd/zones/master/\` `lab.internal.zone` so anpassen, dass die Serial-Nummer (z.B. um 1) erhöht wird und alle SSHFP-Einträge für `s1` einen veränderten Wert haben (z.B. jedes letztes Zeichen ändern)	Zonen-Datei wurde modifiziert

#	Tätigkeit	Ergebnis
5	Auf Server `s1` mittels `nsd-control reload lab.internal` die DNS-Zone neu laden und ggf. den DNS-Resolver auf der Firewall neu starten, um dessen Cache zu leeren (wenn die DNS-Anfragen darüber laufen)	Die veränderten SSHFP-Einträge sind nun aktiv
6	Auf Client `c1` die `known_hosts`-Datei unter `~/.ssh/known_hosts` entfernen	Die Datei ist entfernt

7	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` mit aktiver DNS-SSHFP-Überprüfung einloggen:
	`ssh -o "VerifyHostKeyDNS yes" \`
	`cmd@s1.lab.internal`
		Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich
		Der Host muss zuvor anhand des Fingerprints bestätigt werden, wobei zusätzlich gewarnt wird, dass der Fingerprint, der mittels DNS gefunden wurde, nicht passt
		siehe Quelltext 5.36
		Wäre der Zielhost bereits im `known_hosts` eingetragen, würde dieselbe Meldung ohne Bestätigungsabfrage des Fingerprints erschienen

8	Die Sitzung wird durch Eingabe von `exit` beendet	Sitzung mit `s1` wird beendet
9	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` ohne DNS-SSHFP-Überprüfung einloggen: `ssh cmd@s1.lab.internal`	Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich Der Fingerprint wird nicht mit dem SSHFP-Record geprüft und somit die fehlende Übereinstimmung nicht festgestellt
10	Die Sitzung wird durch Eingabe von `exit` beendet	Sitzung mit `s1` wird beendet
11	Auf Server `s1` die bestehende DNS-Zone für `lab.internal` unter `/var/nsd/zones/master/\` `lab.internal.zone` so anpassen, dass die Serial-Nummer (z.B. um 1) erhöht wird und alle SSHFP-Einträge für `s1` entfernt werden	Zonen-Datei wurde modifiziert und SSHFP-Einträge für `s1` entfernt
12	Auf Server `s1` mittels `nsd-control reload lab.internal` die DNS-Zone neu laden und ggf. den DNS-Resolver auf der Firewall neu starten, um dessen Cache zu leeren (wenn die DNS-Anfragen darüber laufen)	Die veränderten SSHFP-Einträge sind nun aktiv
13	Auf Client `c1` die `known_hosts`-Datei unter `~/.ssh/known_hosts` entfernen	Die Datei ist entfernt

#	Tätigkeit	Ergebnis

14	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` mit aktiver DNS-SSHFP-Überprüfung einloggen:
	`ssh -o "VerifyHostKeyDNS yes" \`
	`cmd@s1.lab.internal`
		Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase und des Benutzerkennworts erfolgreich
		Der Host muss zuvor anhand des Fingerprints bestätigt werden, wobei zusätzlich gemeldet wird, dass der Fingerprint mittels DNS nicht gefunden wurde
		siehe Quelltext 5.37

15	Auf Server `s1` die bestehende DNS-Zone für `lab.internal` unter `/var/nsd/zones/master/\` `lab.internal.zone` so anpassen, dass die alle SSHFP-Einträge für `s1` wieder ihren Anfangswert haben Die ursprüngliche Zonendatei wurde zu Beginn nach `/tmp` kopiert	Zonen-Datei wurde modifiziert
16	Auf Server `s1` mittels `nsd-control reload lab.internal` die DNS-Zone neu laden und ggf. den DNS-Resolver auf der Firewall neu starten, um dessen Cache zu leeren (wenn die DNS-Anfragen darüber laufen)	Die veränderten SSHFP-Einträge sind nun aktiv

Tabelle 5.23.: Verifikation „SSHFP-DNS-Records“ - „Fingerprint in SSHFP-DNS-Record“

```
c1$ ssh -o "VerifyHostKeyDNS yes" cmd@s1.lab.internal
The authenticity of host 's1.lab.internal (192.168.1.1)' can't be established.
ED25519 key fingerprint is SHA256:ctWpch78E8V8qZEv74v72HQaHVTauCgB0xF5xF1OvRU.
Matching host key fingerprint found in DNS.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 's1.lab.internal' (ED25519) to the list of known
   hosts.
Enter passphrase for key '/home/user/.ssh/id_ed25519':
cmd@s1.lab.internal's password:
s1$
```

Quelltext 5.35: Verifikation „SSHFP-DNS-Records“ - „Fingerprint in SSHFP-DNS-Record“ - Verbindungsaufbau zu Host mit passendem SSHFP-DNS-Record

```
c1$ ssh -o "VerifyHostKeyDNS yes" cmd@s1.lab.internal
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    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 a host key has just been changed.
The fingerprint for the ED25519 key sent by the remote host is
SHA256:ctWpch78E8V8qZEv74v72HQaHVTauCgB0xF5xF10vRU.
Please contact your system administrator.
Update the SSHFP RR in DNS with the new host key to get rid of this message.
The authenticity of host 's1.lab.internal (192.168.1.1)' can't be established.
ED25519 key fingerprint is SHA256:ctWpch78E8V8qZEv74v72HQaHVTauCgB0xF5xF10vRU.
No matching host key fingerprint found in DNS.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 's1.lab.internal' (ED25519) to the list of known
   hosts.
Enter passphrase for key '/home/user/.ssh/id_ed25519':
cmd@s1.lab.internal's password:
s1$
```

Quelltext 5.36: Verifikation „SSHFP-DNS-Records“ - „Fingerprint in SSHFP-DNS-Record“ - Verbindungsaufbau zu Host mit unpassendem SSHFP-DNS-Record

```
c1$ ssh -o "VerifyHostKeyDNS yes" cmd@s1.lab.internal
The authenticity of host 's1.lab.internal (192.168.1.1)' can't be established.
ED25519 key fingerprint is SHA256:ctWpch78E8V8qZEv74v72HQaHVTauCgB0xF5xF10vRU.
No matching host key fingerprint found in DNS.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 's1.lab.internal' (ED25519) to the list of known
   hosts.
Enter passphrase for key '/home/user/.ssh/id_ed25519':
cmd@s1.lab.internal's password:
s1$
```

Quelltext 5.37: Verifikation „SSHFP-DNS-Records“ - „Fingerprint in SSHFP-DNS-Record“ - Verbindungsaufbau zu Host ohne SSHFP-DNS-Record

Somit wird aufgezeigt wie mit der zugehörigen SSH-Client-Option die Fingerprints der SSH-Server überprüft werden können und wie sich der Client im Falle unpassender oder keiner SSHFP-DNS-Einträge verhält.

5.9. FIDO2-Authentisierung mit YubiKey

Die FIDO2-Testfälle gehen auf die entsprechende Handhabung und Authentisierung ein, jedoch ohne Anwendung von Zertifikaten oder dem Authentisierungs-Agenten. Mehr zu FIDO2 und Zertifikate ist in Kapitel 4.2.11.2 aufgeführt. Der Versuch mit dem Authentisierungs-Agenten und FIDO2 ist in Kapitel 4.2.11.3 vorzufinden. Es werden ausschliesslich „Resident“-Keys verwendet (auch bekannt als „Discoverable Credentials“, siehe Kapitel 4.2.11).

5.9.1. Schlüssel-Parameter

Dieser Testfall prüft den Zusammenhang zwischen der SSH-Server-Option `PubkeyAuthOptions` und entsprechenden Optionen beim Erstellen des Schlüssels für die Verwendung mit einem FIDO Authenticator.

Vorbedingungen:

- YubiKey
  - Es ist ein YubiKey vorhanden, welcher für den Testfall frei verwendet werden kann (bestenfalls neu oder zurückgesetzt)
  - FIDO2-PIN mittels YubiKey Manager [154] definiert (siehe Kapitel 4.2.11)
- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` ist leer
  - Der YubiKey ist am Client über USB angeschlossen
    In diesem Falle wurde der YubiKey über VirtualBox der Client-VM angehängt, durch Ausführen von `dmesg` nach YubiKey-Anschluss wird ausgegeben, dass dieser erkannt wurde
- Server `s1`
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet
  - In der OpenSSH-Server-Konfiguration ist bei der Option `PubkeyAuthOptions` der Wert `verify-required` hinterlegt

#	Tätigkeit	Ergebnis

1	Auf Client `c1` mit folgendem Befehl einen Ed25519-Schlüssel des Typs `ed25519-sk` gemäss Kapitel 4.2.11.1 lediglich mit der Option `resident` erstellen:
	`ssh-keygen -t ed25519-sk \`
	`-O resident`
		Nach Eingabe des FIDO2-PINs, Berühren des YubiKeys und Hinterlegen einer Key-Passphrase wurde er Key erstellt
		Die Key-Referenzdateien werden zusätzlich unter `~/.ssh/id_ed25519_sk` und `~/.ssh/id_ed25519_sk.pub` abgelegt
		Diese Dateien können jederzeit gelöscht und vom YubiKey gemäss Kapitel 4.2.11.1, Quelltext 4.42 neu ausgelesen werden, werden aber für die nächsten Schritte verwendet

2	Auf Server `s1` unter `/home/cmd/.ssh/authorized_keys` den zuvor generierten Public-Key (`~/.ssh/id_ed25519_sk.pub` auf `c1`) hinterlegen	Public-Key ist hinterlegt

#	Tätigkeit	Ergebnis
3	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase und Berühren des YubiKeys nicht erfolgreich (Berechtigung aufgrund des Public-Keys verweigert) Die Datei `/var/log/authlog` auf `s1` zeigt folgende Meldung: `error: public key ED25519-SK SHA256:4vtod...MCOKA signature for cmd from 192.168.100.1 port 21115 rejected: user verification requirement not met`
4	Auf Server `s1` in der OpenSSH-Server-Konfiguration unter `/etc/ssh/sshd_config` den Wert der Option `PubkeyAuthOptions` auf `touch-required` ändern	Die Konfiguration wird übernommen
5	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase, Berühren des YubiKeys und Eingabe des Benutzerkennworts erfolgreich Die Definition der Server-Option `PubkeyAuthOptions` auf `none` erzielt dasselbe Verhalten siehe Quelltext 5.38
6	Die Sitzung wird durch Eingabe von `exit` beendet	Sitzung mit `s1` wird beendet
7	Auf Server `s1` in der OpenSSH-Server-Konfiguration unter `/etc/ssh/sshd_config` den Wert der Option `PubkeyAuthOptions` zurück auf `verify-required` ändern	Die Konfiguration wird übernommen
8	Auf Client `c1` die Schlüssel-Dateien im Ordner `~/.ssh` löschen	Dateien sind gelöscht
9	Auf Client `c1` mit folgendem Befehl einen neuen Ed25519-Schlüssel des Typs `ed25519-sk` gemäss Kapitel 4.2.11.1 mit den Optionen `resident` und `verify-required` erstellen: `ssh-keygen -t ed25519-sk \ -O resident \ -O verify-required`	Nach Eingabe des FIDO2-PINs, Berühren des YubiKeys und Hinterlegen einer Key-Passphrase wurde er Key erstellt Die Abfrage, ob der bestehende Key überschrieben werden soll, wird bestätigt Die Key-Referenzdateien werden wieder zusätzlich unter `~/.ssh/id_ed25519_sk` und `~/.ssh/id_ed25519_sk.pub` abgelegt

#	Tätigkeit	Ergebnis
10	Auf Server `s1` unter `/home/cmd/.ssh/authorized_keys` den zuvor generierten Public-Key (`~/.ssh/id_ed25519_sk.pub` auf `c1`) hinterlegen und den bestehenden Eintrag überschreiben	Public-Key ist hinterlegt
11	Auf Client `c1` mit folgendem Befehl als Benutzer `cmd` auf dem Server `s1` einloggen: `ssh cmd@192.168.1.1`	Der Login als Benutzer `cmd` ist nach Eingabe der Key-Passphrase, Eingabe des FIDO2-PINs, Berühren des YubiKeys und Eingabe des Benutzerkennworts erfolgreich Die Definition der Server-Option `PubkeyAuthOptions` auf `none` oder `touch-required` erzielt dasselbe Verhalten siehe Quelltext 5.39

Tabelle 5.24.: Verifikation „FIDO2-Authentisierung mit YubiKey“ - „Schlüssel-Parameter“

```
c1$ ssh cmd@192.168.1.1
Enter passphrase for key '/home/user/.ssh/id_ed25519_sk':
Confirm user presence for key ED25519-SK SHA256:4
   vtodSdf87zj0ufTc6cXt4QGk6U7gmPoANto29MCOKA
User presence confirmed
cmd@192.168.1.1's password:
s1$
```

Quelltext 5.38: Verifikation „FIDO2-Authentisierung mit YubiKey“ - „Schlüssel-Parameter“ - Verbindungsaufbau mit FIDO2 und YubiKey

```
c1$ ssh cmd@192.168.1.1
Enter passphrase for key '/home/user/.ssh/id_ed25519_sk':
Confirm user presence for key ED25519-SK SHA256:3PpQsuItrs3JWwIWmW91JPS1W2YD3u
   /diG3OXzxSrgI
Enter PIN for ED25519-SK key /home/user/.ssh/id_ed25519_sk:
Confirm user presence for key ED25519-SK SHA256:3PpQsuItrs3JWwIWmW91JPS1W2YD3u
   /diG3OXzxSrgI
User presence confirmed
cmd@192.168.1.1's password:
s1$
```

Quelltext 5.39: Verifikation „FIDO2-Authentisierung mit YubiKey“ - „Schlüssel-Parameter“ - Verbindungsaufbau mit FIDO2 und YubiKey mit Option `verify-required`

Somit wird aufgezeigt, dass die Option `verify-required` beim Erstellen eines Schlüssels für einen YubiKey nötig ist, sofern in der SSH-Server-Konfiguration bei der Option `PubkeyAuthOptions` der Wert `verify-required` hinterlegt ist. Ist dieser Wert hinterlegt, muss beim Login zusätzlich der FIDO2-PIN eingegeben werden. Bei der Meldung `Confirm user presence` gilt es den YubiKey zu berühren [81].

Zu bemerken ist, dass wie in Kapitel 4.2.11.1 erwähnt die Passwort-Authentisierung mit dieser Konfiguration deaktiviert werden könnte und man trotzdem noch eine Multi-Faktor-Authentisierung durchführen würde.

[81] Diese Meldung erscheint bei der Option `verify-required` stets doppelt, jedoch ist nach dem ersten Mal der PIN einzugeben und beim zweiten Mal der YubiKey zu berühren. Gemäss Client-Debugging-Meldungen scheint der Client den Key zuerst ohne PIN auslesen zu wollen, welcher dann fehlschlägt, dann erst den PIN abfragt und dann nochmals die Berührung anfordert

5.9.2. Mehrere Schlüssel

Anhand dieses Testfalls wird die Option, den Schlüsseln für die Authentisierung mittels FIDO Authenticators einen Identifikator zu verleihen und somit mehrere Schlüssel zu speichern, betrachtet.

Vorbedingungen:

- YubiKey
  - Es ist ein YubiKey vorhanden, welcher für den Testfall frei verwendet werden kann (bestenfalls neu oder zurückgesetzt)
  - FIDO2-PIN mittels YubiKey Manager [154] definiert (siehe Kapitel 4.2.11)
- Client `c1`
  - Das `.ssh`-Verzeichnis des Benutzer `user` auf dem Client `c1` ist leer
  - Der YubiKey ist am Client über USB angeschlossen
    In diesem Falle wurde der YubiKey über VirtualBox der Client-VM angehängt, durch Ausführen von `dmesg` nach YubiKey-Anschluss wird ausgegeben, dass dieser erkannt wurde
- Server `s1`
  - Jumphost-Zugriff mittels Benutzer `jump` gemäss Kapitel 4.2.6, Quelltext 4.11 nicht implementiert
  - In der OpenSSH-Server-Konfiguration ist bei der Option `PubkeyAuthOptions` der Wert `verify-required` hinterlegt
- Server `s2`
  - Kommandozeilenzugriff für Benutzer `cmd` gemäss Kapitel 4.2.3, Quelltext 4.5 eingerichtet
  - In der OpenSSH-Server-Konfiguration ist bei der Option `PubkeyAuthOptions` der Wert `verify-required` hinterlegt

#	Tätigkeit	Ergebnis

1	Auf Client `c1` mit folgendem Befehl einen neuen Ed25519-Schlüssel des Typs `ed25519-sk` gemäss Kapitel 4.2.11.1 mit den Optionen `resident` und `verify-required` sowie dem Identifikator `A` erstellen:
	`ssh-keygen -t ed25519-sk \`
	`-O resident \`
	`-O verify-required \`
	`-O application=ssh:A`
		Nach Eingabe des FIDO2-PINs, Berühren des YubiKeys und Hinterlegen einer Key-Passphrase wurde er Key erstellt
		Die Key-Referenzdateien werden wieder zusätzlich unter `~/.ssh/id_ed25519_sk` und `~/.ssh/id_ed25519_sk.pub` abgelegt

2	Auf Client `c1` nochmals auf die gleiche Art einen Schlüssel erstellen, jedoch mit dem Identifikator `B`	Nach Eingabe des FIDO2-PINs, Berühren des YubiKeys und Hinterlegen einer Key-Passphrase wurde er Key erstellt Die Key-Referenzdateien werden wieder zusätzlich unter `~/.ssh/id_ed25519_sk` und `~/.ssh/id_ed25519_sk.pub` abgelegt und überschreiben die Bestehenden, wobei auch ein anderer Speicherpfad gewählt werden konnte

#	Tätigkeit	Ergebnis
3	Auf Client `c1` nochmals auf die gleiche Art einen Schlüssel erstellen, jedoch mit dem Identifikator `C`	Nach Eingabe des FIDO2-PINs, Berühren des YubiKeys und Hinterlegen einer Key-Passphrase wurde er Key erstellt Die Key-Referenzdateien werden wieder zusätzlich unter `~/.ssh/id_ed25519_sk` und `~/.ssh/id_ed25519_sk.pub` abgelegt und überschreiben die Bestehenden, wobei auch ein anderer Speicherpfad gewählt werden konnte
4	Auf Client `c1` die Schlüssel-Dateien im Ordner `~/.ssh` löschen	Dateien sind gelöscht
5	Auf Client `c1` die Schlüssel-Dateien in den Ordner `~/.ssh` vom YubiKey auslesen: `cd ~/.ssh && ssh-keygen -K`	Nach Angabe einer Key-Passphrase werden die Key-Referenzdateien neu ausgelesen und anhand ihres Identifikators abgespeichert Die eingegebene Passphrase wird für alle Referenzdateien angewendet, wobei beim nächsten Auslesen wieder eine neue Passphrase gesetzt werden kann siehe Quelltext 5.40
6	Auf Server `s1` unter `/home/jump/.ssh/authorized_keys` den zuvor generierten Public-Key mit Identifikator `A` (`~/.ssh/id_ed25519_sk_rk_A.pub` auf `c1`) hinterlegen und den bestehenden Eintrag überschreiben	Public-Key ist auf Server `s1` hinterlegt
7	Auf Server `s2` unter `/home/cmd/.ssh/authorized_keys` den zuvor generierten Public-Key mit Identifikator `B` (`~/.ssh/id_ed25519_sk_rk_B.pub` auf `c1`) hinterlegen und den bestehenden Eintrag überschreiben	Public-Key ist auf Server `s2` hinterlegt

8	Auf Client `c1` mit folgendem Befehl als Benutzer `jump` den Server `s1` als Jumphost genutzt und darüber als Benutzer `cmd` auf Server `s2` mit Angabe zugehöriger Schlüssel eingeloggt:
	`ssh -v -i id_ed25519_sk_rk_A \`
	`-i id_ed25519_sk_rk_B \`
	`-J jump@192.168.1.1 \`
	`cmd@192.168.2.1`
		Der Login wird aufgrund des Public-Keys verweigert, wobei aus der Ausgabe ersichtlich ist, dass keiner der YubiKey-Referenzdateien verwendet wurde
		siehe Quelltext 5.41

9	Auf Client `c1` in der SSH-Client-Konfigurationsdatei `~/.ssh/config` gemäss der Manpage `ssh_config` [57] mit der Option `IdentityFile` die Schlüssel für die entsprechenden Zielhosts referenzieren und zusätzliche Parameter zum vorherigen Jumphost-Versuch einbinden siehe Quelltext 5.42	Die Konfiguration wurde angelegt

#	Tätigkeit	Ergebnis
10	Auf Client `c1` mit folgendem Befehl als Benutzer `jump` den Server `s1` als Jumphost genutzt und darüber als Benutzer `cmd` auf Server `s2` mit Angabe zugehöriger Schlüssel eingeloggt: `ssh -v 192.168.2.1` Die Parameter für die Verbindung zum Zielhost wird nun aus der zuvor angelegten Client-Konfigurationsdatei (Quelltext 5.42) ausgelesen	Der Login wird nach ▶ Angabe der Passphrase für den Key mit Identifikator `A` ▶ Angabe des FIDO2-PINs für den Key mit Identifikator `A` ▶ Berühren des YubiKeys für den Key mit Identifikator `A` ▶ Angabe des Benutzerkennworts für Benutzer `jump` auf Server `s1` ▶ Angabe der Passphrase für den Key mit Identifikator `B` ▶ Angabe des FIDO2-PINs für den Key mit Identifikator `B` ▶ Berühren des YubiKeys für den Key mit Identifikator `B` ▶ Angabe des Benutzerkennworts für Benutzer `cmd` auf Server `s2` durchgeführt siehe Quelltext 5.43

Tabelle 5.25.: Verifikation „FIDO2-Authentisierung mit YubiKey“ - „Mehrere Schlüssel“

```
c1$ cd ~/.ssh && ssh-keygen -K
Enter PIN for authenticator:
You may need to touch your authenticator to authorize key download.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Saved ED25519-SK key ssh:A to id_ed25519_sk_rk_A
Saved ED25519-SK key ssh:B to id_ed25519_sk_rk_B
Saved ED25519-SK key ssh:C to id_ed25519_sk_rk_C
c1$ ls -l
total 24
-rw-------  1 user  user  496 Sep 15 17:15 id_ed25519_sk_rk_A
-rw-r--r--  1 user  user  134 Sep 15 17:15 id_ed25519_sk_rk_A.pub
-rw-------  1 user  user  496 Sep 15 17:15 id_ed25519_sk_rk_B
-rw-r--r--  1 user  user  134 Sep 15 17:15 id_ed25519_sk_rk_B.pub
-rw-------  1 user  user  496 Sep 15 17:15 id_ed25519_sk_rk_C
-rw-r--r--  1 user  user  134 Sep 15 17:15 id_ed25519_sk_rk_C.pub
```

Quelltext 5.40: Verifikation „FIDO2-Authentisierung mit YubiKey“ - „Mehrere Schlüssel“ - Auslesen von Schlüssel-Referenzdateien von einem YubiKey

```
c1$ ssh -v -i id_ed25519_sk_rk_A -i id_ed25519_sk_rk_B -J jump@192.168.1.1
    cmd@192.168.2.1
...
debug1: identity file id_ed25519_sk_rk_A type 12
debug1: identity file id_ed25519_sk_rk_A-cert type -1
debug1: identity file id_ed25519_sk_rk_B type 12
debug1: identity file id_ed25519_sk_rk_B-cert type -1
...
debug1: Authenticating to 192.168.1.1:22 as 'jump'
...
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Trying private key: /home/user/.ssh/id_rsa
debug1: Trying private key: /home/user/.ssh/id_ecdsa
debug1: Trying private key: /home/user/.ssh/id_ecdsa_sk
debug1: Trying private key: /home/user/.ssh/id_ed25519
debug1: Trying private key: /home/user/.ssh/id_ed25519_sk
debug1: Trying private key: /home/user/.ssh/id_xmss
debug1: Trying private key: /home/user/.ssh/id_dsa
debug1: No more authentication methods to try.
jump@192.168.1.1: Permission denied (publickey).
```

Quelltext 5.41: Verifikation „FIDO2-Authentisierung mit YubiKey“ - „Mehrere Schlüssel“ - Verbindungsversuch mit Angabe bestimmter, unüblich benannter Schlüssel

```
Host 192.168.1.1
    IdentityFile id_ed25519_sk_rk_A

Host 192.168.2.1
    IdentityFile id_ed25519_sk_rk_B
    ProxyJump jump@192.168.1.1
    User cmd
```

Quelltext 5.42: Verifikation „FIDO2-Authentisierung mit YubiKey“ - „Mehrere Schlüssel“ - SSH-Client-Konfiguration mit Angabe bestimmter, unüblich benannten Schlüssel und Jumphost-Konfiguration für zu Server `s2` via `s1`

```
c1$ ssh -v 192.168.2.1
...
debug1: Authenticating to 192.168.1.1:22 as 'jump'
...
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Offering public key: id_ed25519_sk_rk_A ED25519-SK SHA256:6
    DlWX9dOYhCVPRPoIDsNKncUWCf/ZdWYmHo9nvjZRm8 explicit authenticator
debug1: Server accepts key: id_ed25519_sk_rk_A ED25519-SK SHA256:6
    DlWX9dOYhCVPRPoIDsNKncUWCf/ZdWYmHo9nvjZRm8 explicit authenticator
Enter passphrase for key 'id_ed25519_sk_rk_A':
...
Enter PIN for ED25519-SK key id_ed25519_sk_rk_A:
Confirm user presence for key ED25519-SK SHA256:6DlWX9dOYhCVPRPoIDsNKncUWCf/
    ZdWYmHo9nvjZRm8
...
User presence confirmed
Authenticated using "publickey" with partial success.
debug1: Authentications that can continue: password
debug1: Next authentication method: password
jump@192.168.1.1's password:
Authenticated to 192.168.1.1 ([192.168.1.1]:22) using "password".
...
debug1: Authenticating to 192.168.2.1:22 as 'cmd'
...
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Offering public key: id_ed25519_sk_rk_B ED25519-SK SHA256:21
    ySzFuGKWo532Pd1F3VQleZ9LK2t/j3H/T5SAhkWk0 explicit authenticator
debug1: Server accepts key: id_ed25519_sk_rk_B ED25519-SK SHA256:21
    ySzFuGKWo532Pd1F3VQleZ9LK2t/j3H/T5SAhkWk0 explicit authenticator
Enter passphrase for key 'id_ed25519_sk_rk_B':
...
Enter PIN for ED25519-SK key id_ed25519_sk_rk_B:
Confirm user presence for key ED25519-SK SHA256:21ySzFuGKWo532Pd1F3VQleZ9LK2t/
    j3H/T5SAhkWk0
..
User presence confirmed
Authenticated using "publickey" with partial success.
debug1: Authentications that can continue: password
debug1: Next authentication method: password
cmd@192.168.2.1's password:
Authenticated to 192.168.2.1 (via proxy) using "password".
...
s2$
```

Quelltext 5.43: Verifikation „FIDO2-Authentisierung mit YubiKey“ - „Mehrere Schlüssel“ - Verbindungsaufbau mit Verwendung der Client-Konfiguration von Quelltext 5.42 (Jumphost und mehrere, unüblich benannter Schlüssel)

Hiermit wird gezeigt, wie mehrere Schlüssel mit einem FIDO Authenticator (hier ein YubiKey) verwendet werden können. Der am Ende von Kapitel 5.5.2 referenzierte Fall, dass bei der Verwendung mehrerer, nicht standardmässig benannter Schlüssel eine Client-Konfigurationsdatei nötig ist, wird hier ebenfalls aufgezeigt.

5.10. Interpretation / Überblick

Mit zuvor durchgeführten Testfällen konnten der Aufbau, die Datenflüsse und das Verhalten ausgewählter OpenSSH-Funktionalitäten verifiziert werden.

Folgende Punkte sind hervorzuheben:

- Die Qualität der OpenSSH-Software konnte aus erster Hand geprüft werden und macht einen hochwertigen Eindruck
- Ein Benutzer mit Kommandozeilenzugriff verfügt über die meisten Möglichkeiten
  Er kann Dateien übertragen (auch ohne SFTP) und je nach Berechtigungen Änderungen am System vornehmen
  siehe u. a. Kapitel 5.2.2
- Der Kommandozeilenzugriff lässt sich durch Hinterlegung auszuführender Befehle einschränken
  siehe Kapitel 5.2.3
- Eine weitere Einschränkungsmöglichkeit bietet das Konfigurieren zulässiger oder unzulässiger Benutzer bzw. Gruppen
  siehe Kapitel 5.1.4
- Local oder Remote Forwarding ermöglichen neue Datenflüsse, welche zu Überwachen und Regulieren sind
  siehe Kapitel 5.1.6
- Reine Dateiübertragungen mittels SFTP können auf gewünschte Pfade eingeschränkt werden und benötigen keinen Kommandozeilenzugriff
  siehe Kapitel 5.3
- Jumphosts bieten die Möglichkeit, SSH-Zugänge (möglichst) zentral zu definieren und loggen
  siehe Kapitel 5.5
- Verbindungen zu älteren Zielhosts bedürfen an Parameter-Anpassungen am SSH-Client (auch für Verbindungen über Jumphosts)
  siehe Kapitel 5.5.2
- Der Authentisierungs-Agent erlaubt das Verwenden von Keys auf einem Server, die alleine dem Client vorliegen
  Zudem können Keys beim Einbinden in den Agenten eingeschränkt werden
  siehe Kapitel 5.6
- Zertifikate lösen die Notwendigkeit, Public-Keys auszutauschen, ab
  Dafür wird das Vertrauen in eine CA gelegt, die entsprechend zu schützen ist
  Je nach angewandten Parametern bei den Zertifikaten wird die Konfiguration der Infrastruktur für SSH-Verbindungen vereinfacht, kann aber auch komplizierter ausfallen [82]
  siehe Kapitel 5.7
- SSH-Server können durch den Einsatz von Zertifikaten und/oder SSHFP-DNS-Einträgen [83] dem Client vertrauenswürdiger erscheinen, sofern die Clients diese Aspekte beim Verbindungsaufbau auch überprüfen
  siehe Kapitel 5.7.3 und 5.8.1
- Die Authentisierung mit einem FIDO Authenticator (hier ein YubiKey) erlaubt das Ablösen von Passwörtern und kann bei Bedarf neben der Berührung des YubiKeys mit einem PIN zusätzlich abgesichert werden
  Die PIN-Abfrage kann durch die Konfiguration des SSH-Servers forciert werden
  siehe Kapitel 5.9

[82] z.B. sind zusätzlich die Principals in einem Zertifikat oder dessen Ablaufdatum zu prüfen. Das Verwenden von „Revocation Listen“ würde weitere Sicherheit, aber auch Komplexität mit sich bringen

[83] bestenfalls abgesichert mittels DNSSEC

6. Abschluss

6.1. Fazit

Der **SSH-Serverdienst von OpenSSH** ist nach der Auseinandersetzung mit dem SSH-Protokoll und dessen Parametern greifbarer und ermöglicht damit einen sicheren Umgang. Die Software befindet sich **nahe am Stand der Technik** und wird generell schnell gegen bekannte Schwachstellen aktualisiert. Sie **bietet mit ihrer Standardkonfiguration einen Kompromiss zwischen Sicherheit und Komfort, welche es weiter abzusichern gilt**, um erlaubte Tätigkeiten wie Kommandozeilenzugriffe oder Forwarding weiter einzuschränken und die Authentisierung nicht nur auf das Benutzerkennwort zu stützen. Besonderen Fokus ist auf den Kommandozeilenzugriff zu legen, mit welchem auch unabhängig von SFTP Dateien übertragen und je nach Berechtigungen Systemanpassungen inkl. der SSH-Server-Konfiguration vorgenommen werden können.

Weitere Absicherungsmöglichkeiten werden durch die Zertifikatsauthentisierung, das Verwenden von FIDO2 sowie das Anlegen einer durchdachten SSH-Server-Konfiguration (mit den nötigsten Berechtigungen und angewandten Empfehlungen von diversen Publikationen und Behörden) geboten. Diese Möglichkeiten können **in diversen Ausprägungen**, je nach Bedarf, implementiert werden, um z.B. bei Zertifikaten Ablaufdaten oder „Revocation Listen“ zu verwenden oder bei einem YubiKey zusätzlich den PIN abzufragen. Die **Algorithmen-Wahl der Standardkonfiguration des OpenSSH-Serverdienstes wird begründet getroffen und möglichst aktuell gehalten, entspricht jedoch nicht jeden Vorgaben, die in der Praxis anzutreffen sind**. Je nach Vorgaben gilt es diese Auswahl manuell zu pflegen und regelmässig zu prüfen.

Vertrauen in SSH-Server können durch Host-Zertifikate von vertrauenswürdigen CAs und/oder SSHFP-DNS-Einträge gefestigt werden. Hierbei gilt es, dass der SSH-Client diese auch überprüfen muss, was bei den Host-Zertifikaten durch Hinterlegung des CA-Public-Keys geschieht, für SSHFP-Überprüfungen jedoch bei jedem Verbindungsaufbau explizit angegeben werden muss.

Mittels Local und/oder Remote Forwarding können Datenflüsse über den OpenSSH-Server eingerichtet werden, womit dieser z.B. als Jumphost verwendet werden kann, um über diesen auf weitere SSH-Server zugreifen zu können. **Die Jumphost-Funktionalität bietet gemäss SSH-Client-Manpage eine sicherere Alternative gegenüber dem Einsatz von Agent-Forwarding**, wobei der Einsatz eines Authentisierungs-Agenten ohne Agent-Forwarding die Anzahl der Eingabe der Key-Passphrase minimiert.

Unabhängig von den konfigurierbaren Absicherungen ist der betriebene **Host (Client und Server) stets aktuell zu halten**, um gegen bekannte und behobene Schwachstellen geschützt zu sein. Dies gilt nicht nur für die OpenSSH-Software auf dem Host. Zusätzlich eingerichtete Serverdienste sind ebenfalls zu prüfen und entsprechend abzusichern. **Definierte Arbeitsflüsse** für unterschiedliche Szenarien **erleichtern den Umgang** und vermindern die Wahrscheinlichkeit eines menschlichen Fehlers, bei welchem etwas falsch angepasst oder vergessen wird.

Zusammenfassend bietet diese Arbeit einen Überblick über die Vielzahl an Möglichkeiten mit dem SSH-Serverdienst von OpenSSH, welche zu dessen Absicherung oder Funktionserweiterung verwendet werden können. Je nach Anforderungen kann die Komplexität schnell zunehmen. **Die in dieser Arbeit ermittelte Grundkonfiguration und Erweiterungen für bestimmte Anwendungsfälle** (Kommandozeilenzugriff, Dateiübertragungen und Jumphosts) **bieten einen möglichst sicheren Aufbau mit einer Multi-Faktor-Authentisierung, ohne eine allzu hohe Komplexität einzuführen**. Anstelle der Passwort-Authentisierung kann ein YubiKey mit aktivierter PIN-Abfrage zusammen mit der Public-Key-Authentisierung verwendet werden.

6.2. Rückblick

Die Arbeit stellte sich als äusserst lehrreich und lohnenswert, aber auch sehr umfassend heraus. Einzelne Parameter der SSH-Server-Konfiguration können stark vertieft werden, was schnell zusätzlichen Aufwand mit sich bringt. Aufgrund des hohen Interesses und dem Bedürfnis, eine möglichst umfassende Übersicht zu bieten, wurde entsprechend mehr Zeit in die Arbeit investiert. Trotzdem gab es Punkte wie zum Beispiel das Handhaben von Zertifikaten mit einem YubiKey, welche aus Zeitgründen nicht weiter verfolgt wurden. Bei diesen Punkten wurde vor dem Verzicht zur weiteren Vertiefung die Komplexitätsstufe geprüft, wobei mit der Steigung der Komplexität die Wahrscheinlichkeit zum alltäglichen Einsatz der Option sinkt.

Im Laufe der Arbeit wurde eine neue OpenSSH-Version (9.4) veröffentlicht [63], wobei diese im aktuellsten OpenBSD-Release (7.3) nicht zur Verfügung steht, da bestehende OpenBSD-Releases keine neuen Funktionalitäten sondern lediglich Updates zur Sicherheit und Stabilität erhalten [157][158]. Aus diesem Grund sowie Zeitgründen wurde der Entscheid getroffen, für diese Arbeit OpenSSH in Version 9.3 weiter zu verwenden. Die Veröffentlichung der neuen Version macht einem zudem nochmals bewusst, seine Infrastruktur auf dem aktuellsten Stand zu halten und regelmässig die Funktionalitäten neuer Software zu prüfen.

Der zu Beginn in der Arbeit erwähnte **Vergleich zwischen Standardausführungen und Versionen diverser Produkte** [84] **wurde aus Zeitgründen nicht durchgeführt, könnte jedoch aufgrund der aus dieser Arbeit gewonnenen Erkenntnissen gut selbstständig durchgeführt werden**. So könnte pro Ausführung dessen SSH-Server-Konfiguration mittels Parameter `-T` [85] getestet und so ausgegeben werden, dass diese sich einfach vergleichen lassen. Die Bedeutung der Optionen und entsprechenden Werten kann u. a. dieser Arbeit entnommen werden, um die Unterschiede zu ermitteln und interpretieren zu können.

Gerne wäre ich noch weiter auf die im Ausblick erwähnten Themen eingegangen, bin im Gesamten jedoch mit der Arbeit sehr zufrieden. **Die ermittelte Grundkonfiguration sowie Erweiterungsmöglichkeiten werden mit hoher Sicherheit zukünftig verwendet**, wobei die Begründung für die Wahl der Parameter ebenfalls in der Arbeit nachgeschlagen werden kann. Das Wissen zu SSH und zum aktuellen Stand der Technik bezüglich empfohlenen Algorithmen konnte weiter gefestigt sowie einem kleinen Exkurs zu Diskussionen über gegen Quantencomputer resistente Algorithmen unternommen werden.

Durch die nun gewonnenen Erkenntnisse über OpenSSH habe ich die Software noch mehr zu schätzen gelernt und werde sie nun noch sicherer einsetzen können.

An dieser Stelle möchte ich mich bei meinem Experten Hansjürg Wenger für das Lesen vorgängiger Versionen und Besprechungen zu dieser Arbeit bedanken.

[84] siehe Listenpunkt 6 in Kapitel 2.1
[85] siehe Tabelle 3.3 in Kapitel 3.4.3.1

6.3. Ausblick

Die OpenSSH-Software bietet trotz einer ausführlichen Auseinandersetzung weitere Gebiete, die nach dieser Arbeit betrachtet werden können.

Eines der Gebiete ist definitiv das **Logging** des OpenSSH-Serverdienstes **sowie das Auswerten entsprechender Logs**, um u. a. Verbindungsversuche von potentiellen Angreifern detektieren zu können. Die **SSH-Client-Konfiguration** kann ebenfalls genauer analysiert werden, um den SSH-Verbindungsaufbau komfortabler zu gestalten, vor allem bei einer erhöhten Komplexität. Ein **automatisierter Aufbau einer Infrastruktur mit Bezug zur OpenSSH-Server- und -Client-Konfiguration** könnte weitere Sicherheit mit einer entsprechenden Komplexitätserhöhung mit sich bringen, welche im alltäglichen Gebrauch weniger spürbar sein könnte.

Diverse Authentisierungsmethoden wie die Benutzerauthentisierung mittels GSS-API, „Keyboard-Interactive"-Authentisierung oder Authentisierung mittels Kerberos könnten weiter vertieft werden, um zum Beispiel **entfernte Benutzer auf dem SSH-Server** verwenden zu können. Die Handhabung von **Zertifikaten sowie „Revocation Listen"** bietet mit dessen Parametern eine weitere Vielzahl an Absicherungsmöglichkeiten. Das **Forwarding von Unix Sockets** und Konfigurieren von **Tunneling** stellen weitere zu untersuchende Funktionalitäten dar.

Diese Punkte bleiben offene Fragen, dessen Beantwortung lohnenswert erscheint.

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Glossar

3DES (Triple Data Encryption Standard) (auch TDES, TDEA)

Symmetrischer Verschlüsselungsalgorithmus (3DES gemäss NIST nach 2023 nicht mehr erlaubt [20]). 7

AEAD (Authenticated Encryption With Associated Data)

Blockchiffre Betriebsart, welche neben der Vertraulichkeit auch die Authentizität und Integrität sicherstellt. Üblicherweise eine Kombination aus Verschlüsselung und MAC. 42

AES (Advanced Encryption Standard)

Symmetrischer Block-Verschlüsselungsalgorithmus mit fixer Blocklänge von 128 Bit und Schlüssellängen von 128, 192 oder 256 Bit. 7, 22, 26, 42, 160

AES-CCM (Counter with CBC-MAC)

Kombination von AES-CTR mit CBC-MAC [159]. 42

AES-GCM (Galois/Counter Mode)

Kombination von AES-CTR mit GHASH MAC [160][161]. 26, 42, 97

BSD (Berkeley Standard Distribution) [162]

Auf Unix basierendes Betriebssystem, Begriff wird heute für Gruppe von Betriebssystemen auf BSD-Basis verwendet (u.a. FreeBSD, NetBSD, OpenBSD). 1, 164

CA (Certificate Authority)

Instanz, welche digitale Zertifikate herausgibt. ii, 5, 22, 26, 30, 50, 55, 71–74, 76, 77, 85, 87–89, 92, 97, 98, 100, 130–135, 149, 150, 156, 157, 177, 183

CBC (Cipher Block Chaining)

Blockchiffre Betriebsart, in welchem der vorgängig verschlüsselte Block für die Verschlüsselung des nächsten Blocks verwendet wird. 7, 42, 160

ChaCha20-Poly1305

ChaCha20 Verschlüsselungsalgorithmus kombiniert mit Poly1305 MAC [163]. 26, 50

Chosen-plaintext Attacke

Angriffsmodell, bei welchem der Angreifer Ciphertexte zu beliebigen Klartexten erhält. 7

CTR (Counter)

Blockchiffre Betriebsart, in welchem der Schlüssel-Stream u. a. mit einem Counter generiert wird und den Block-Cipher in einen Stream-Cipher verwandelt. 7, 26, 42, 160

Curve25519

Spezifische elliptische Kurve. 13, 26, 56, 71, 83, 162, 167

Curve448 (auch Curve448-Goldilocks)

Spezifische elliptische Kurve. 13, 162, 167

CVE (Common Vulnerabilities and Exposures)

Standard zur Verwaltung von Schwachstellen [99]. 47–49, 154

CVSS (Common Vulnerability Scoring System)

Standard zur Bewertung des Schweregrades von Schwachstellen mit einer Skala von 0.0 bis 10.0 [98]. 47–49

CWE (Common Weakness Enumeration)

Liste von Soft- und Hardware-Verwundbarkeitstypen [138]. 48

Daemon

Programm, welches als Hintergrundprozess betrieben wird. 15, 23, 34, 35, 39, 78, 80

DCSP (Differentiated Services Code Points)

Punkte für DiffServ (Schema zur Klassifizierung von IP-Paketen). 35

Debian

Linux-Distribution. 3, 43, 120, 121, 124

Diffie–Hellman Key Exchange

Protokoll zur Vereinbarung eines gemeinsamen Schlüssels mit Beziehung zum Diskreten Logarithmus. 4, 7, 8, 13, 24, 26, 42, 44

DLIES (Discrete Logarithm Integrated Encryption Scheme)

Hybrides Verschlüsselungsverfahren (Kombination aus asymmetrischen Verfahren für den Versand eines symmetrischen Schlüssels, mit welchem eine Nachricht verschlüsselt wird) mit Beziehung zum Diskreten Logarithmus. 41

DNSSEC (Domain Name System Security)

Erweiterung des DNS (Domain Name System) um Sicherheitsmechanismen. 5, 13, 80, 89, 91, 149

DoS (Denial of Service) Attacke

Angriff, bei welchem viele gezielte Anfragen eine Blockierung des Dienstes verursachen. 4, 50

DSA (Digital Signature Algorithm)

Asymmetrisches kryptografisches Verfahren mit Beziehung zum Diskreten Logarithmus. 5, 8, 42, 116, 117, 158

DSS (Digital Signature Standard)

Standard zur Spezifizierung von Algorithmen zur Generierung digitaler Signaturen. 5, 8, 98

ECC (Elliptic Curve Cryptography)

Kryptografie basierend auf den Strukturen elliptischer Kurven (spezielle algebraische Kurve). 8, 13

ECDH (Elliptic Curve Diffie-Hellman) key exchange

DHKE basierend auf elliptischen Kurven. 8, 13, 26, 42, 167, *siehe* Diffie–Hellman Key Exchange

ECDSA (Elliptic Curve Digital Signature Algorithm)

Variante von DSA, nutzt die Elliptische-Kurven-Kryptografie oder Elliptic Curve Cryptography (ECC). 5, 8, 13, 22, 26, 28, 29, 36, 42, 57, 78, 83, 181

ECIES (Elliptic Curve Integrated Encryption Scheme)

Hybrides Verschlüsselungsverfahren (Kombination aus asymmetrischen Verfahren für den Versand eines symmetrischen Schlüssels, mit welchem eine Nachricht verschlüsselt wird) mit Gebrauch von elliptischen Kurven. 41

ECMQV (Elliptic Curve Menezes-Qu-Vanstone) key exchange

Protokoll zur Vereinbarung eines gemeinsamen Schlüssels basierend auf elliptischen Kurven. 8

Ed25519

EdDSA mit SHA-512 und Curve25519 [164]. 8, 13, 22, 23, 26, 28, 29, 36, 42, 56, 57, 71, 73, 78, 83, 93, 95, 97, 102–105, 108, 109, 111, 112, 114, 115, 118, 120–122, 125–130, 132, 134, 137, 141, 142, 144, 156–158, 181

Ed448

EdDSA mit SHA-3-Funktion „SHAKE256“ und Curve448 [164]. 8, 13, 42

EdDSA (Edwards-curve Digital Signature Algorithm)

Variante der Schnorr-Signatur (Verfahren mit Beziehung zum Diskreten Logarithmus) auf Basis von „twisted Edwards Kurven“ (Familie von Ellipitischen Kurven) [164]. 5, 42, 56, 71, 83

FIDO Authenticator

Kryptografische Entität in Hard- oder Software, mit welcher ein Benutzer registriert und z.B. mit dem zugehörigen Public-Key bescheinigt werden kann [165]. 20, 29, 39, 61, 83, 85, 86, 89, 141, 144, 148, 149, 157, 162

FIDO2/WebAuthn

Kryptographisches „Request-Response“-Protokoll zur Authentisierung zwischen einem „Relying Party“-Server, dem Client und einem FIDO Authenticator (z.B. in Form eines YubiKeys) [165]. ii, 82–85, 87–90, 141–146, 150, 159, 177

Fingerprint (auch Thumbprint)

Identifikator eines Public-Keys, erstellt durch eine darauf applizierte Hash-Funktion. 13, 20, 26, 39, 78, 80, 87, 89–91, 95, 96, 135, 137–140

FQDN (Fully Qualified Domain Name)

Absoluter Name einer Domain in der Baum-Hierarchie des Domain Name Systems (DNS). 5, 137

FTP (File Transfer Protocol)

Kommunikationsprotokoll zur Übertragung von Dateien zwischen Server und Client über ein Netzwerk. 19

GSS-API (Generic Security Services Application Program Interface)

Programmierschnittstelle für Anwendungen zum Zugriff auf Security Devices, erlaubt es z.B. Kerberos-Implementationen API-kompatibel zu machen. 13, 27, 152

HMAC (Hash-based Message Authentication Code)

MAC-Algorithmus. 7, 13, 26, 42

Hypervisor

Software, Firmware oder Hardware, welche die Erzeugung und den Betrieb von Virtuellen Maschinen (VMs) ermöglicht. 3

IANA (Internet Assigned Numbers Authority)

Behörde für die Zuordnung von Namen und Nummern im Internet. 5

IETF (Internet Engineering Task Force)

Organisation zur Standardisierung des Internets. 5

Kerberos

Netzwerk-Authentisierungsprotokoll. 13, 27, 152, 163

Kerberos Key Distribution Center (KDC)

Entität, welche Benutzer mittels Kerberos verifiziert und authentisiert. 27

KEX (Key Exchange)

Schlüsselaustausch, siehe Kapitel 3.1.3.5. 26, 98, 100, 157

Key

Schlüssel. 5, 13, 17, 20, 23, 24, 29, 39, 68–70, 77, 90, 91, 95, 98, 100, 102–104, 108, 110–115, 118, 121–123, 125–129, 131–133, 135, 137–139, 141–146, 149, 150, 156–158, 164, *siehe* Schlüssel

LibreSSL

Programmbibliothek und Programme zur Anwendung von Transport Layer Security (TLS) und Kryptographie, „Fork“ von OpenSSL von 2014 [166]. 49

Local Forwarding

Weiterleitung eines Port von Client zu Server, wobei der Client auf einem Port hört und zugehörige Verbindungen an den Server weiterleitet [69]. 31, 32, 45, 50, 66, 105, 106, 150, 153, 154, 156, 158, 176

MAC (Message Authentication Code)

Mit Schlüssel parametrisierte Hash-Funktion. 5–7, 9, 17, 24, 26, 97, 99, 100, 157, 160, 163, 167

Man-in-the-Middle-Angriff

Angriff bei dem der Angreifer sich physisch oder logisch zwischen zwei Komunikationspartnern befindet und somit die Kontrolle über den entsprechenden Datenverkehr hat. 5, 18, 21

Manpage

kurz für „manual page“. Dokumentation, die bei Unix- oder Unix-ähnlichen Betriebssystemen mit dem Befehl `man` angezeigt werden kann. 18, 20, 24, 26, 27, 30, 33, 37–39, 46, 50, 61, 67, 69, 70, 73, 124, 145, 150

NIST (National Institute of Standards and Technology)

Bundesbehörde der USA, für Standardisierungsprozesse zuständig, entwickelte u. a. den SHA-Hashalgorithmus und gab u. a. den Verschlüsselungsalgorithmus AES bekannt. 7, 160

OAEP (Optimal Asymmetric Encryption Padding)

Kryptografisches Padding-Verfahren. 42

OpenBSD

Betriebssystem auf Basis von BSD Version 4.4 (bekannt als „4.4BSD“) [5]. ii, 1, 3, 15, 24, 36, 43, 49, 51, 54, 57, 63, 67, 71, 78, 86, 120, 151, 165, 176

OpenSSH

Software-Implementation des SSH-Protokolls [46]. ii, 1–5, 13, 15, 19, 22, 23, 26, 44, 46–50, 55, 58, 73, 82, 86, 87, 120, 121, 124, 135, 141, 142, 144, 149–152, 154, 159, 166, 176, 179, 185, 186

PEM (Privacy-Enhanced Mail)

Format zur Handhabung von Schlüsseln, Zertifikaten und anderen Daten. 22

Perfect Forward Secrecy

Eigenschaft in der Kryptografie für Schlüsselaustauschprotokolle, wenn die Kompromittierung eines Session-Keys keinen Einfluss auf eine andere Session hat oder wenn die Kompromittierung eines Langzeit-Schlüssels keine vergangenen Session-Keys kompromittiert [167]. 4

pfSense

Firewall / Router Software Distribution [8]. 3, 51, 53

PKCS #11 (Public Key Cryptography Standard #11: Cryptographic Token Interface, auch PKCS#11)

Standard zur Public-Key-Kryptografie zur Definition einer Schnittstelle zu kryptografischen Tokens wie z.B. Smartcards. 20, 39

PKCS #8 (Public Key Cryptography Standard #8: Private-Key Information Syntax Standard, auch PKCS#8)

Standard zum Beschrieb von Private-Keys und Public-Keys. 22

Principal

Bezeichnung, welche bei der Zertifikatserstellung zu einem Schlüssel (z.B. mit `ssh-keygen -n` [53]) mitgegeben werden kann, wobei die Angabe mehrerer „Principals“ möglich sind [68]. 30, 72, 73, 97, 130–134, 149, 156, 181, 183

Private-Key

Geheimer Schlüssel, der geheim bleiben muss. 5, 11, 15, 17, 20, 22, 23, 36, 39, 40, 43, 56, 57, 67, 70, 71, 74, 75, 82–87, 156, 157, 164, 176, *siehe* Schlüssel

Pseudo-Terminal

Paar von Geräten, bei welchem die Eingaben ins erste Gerät dem Zweiten übergeben werden und wiederum Ausgaben auf dem zweiten Gerät dem Ersten übergeben werden [168][169]. 12, 24, 36, 37, 61, 62

Public-Key

Öffentlicher Schlüssel, der jedem zugänglich sein muss. 5, 6, 8–11, 13, 15, 17, 18, 20–24, 26, 27, 29, 30, 36, 39, 43, 44, 46, 56, 57, 61–63, 65–68, 70–78, 83–91, 95–98, 101–105, 108, 109, 111, 112, 114–116, 118, 120–122, 124–135, 137, 141–143, 145, 149, 150, 156–158, 162, 164, 181, *siehe* Schlüssel

RC-Skript

Skript, welches bestimmte Initialisierungsroutinen enthaltet. 24, 36, 37, 61

rdomain (Routing Domain)

Eine Routing Domäne erlaubt das logische Aufteilen eines Routers im OpenBSD-Kernel, wobei eine rdomain einem separierten Adressbereich im Speicher des Kernels entspricht. Eine IP-Adresse kann mehreren rdomains zugewiesen werden, aber nur ein mal pro rdomain. Ein Netzwerkinterface gehört dabei nur zu einer einzelnen rdomain, welche mindestens eine Routing Tabelle hat [170]. 35, 165

Remote Forwarding

Weiterleitung eines Port von Server zu Client, wobei der Server auf einem Port hört und zugehörige Verbindungen an den Client weiterleitet, der die Verbindung an einen hinterlegten Port sendet, um z.B. jemandem von Aussen Zugriff auf einen interne Webseite zu ermöglichen [69]. 31, 32, 34, 45, 50, 105, 107, 149, 150, 153, 154, 158, 176

Replay Attacke

Angriff, bei welchem legitime Übertragungen wiederholt werden. 6

Revocation List

Liste mit revozierten bzw. für ungültig erklärte Zertifikate oder Schlüssel. 22, 29, 73, 90, 91, 149, 152

RFC (Request for Comments)

Dokumente zur technischen Spezifizierung und organisatorischen Notizen zum Internet. 7, 8, 11, 13, 14, 19, 22, 40, 41, 176

RSA (Rivest–Shamir–Adleman)

Asymmetrisches kryptografisches Verfahren mit Beziehung zum Faktorisierungsproblem. ii, 5, 8, 13, 22, 26–29, 36, 42, 44, 57, 73, 78, 116, 117, 122, 124, 158, 181

Schlüssel

Information, die einen kryptografischen Algorithmus parametrisiert (im einfachsten Fall ein Kennwort), um z.B. einen Text zu ver- oder entschlüsseln. ii, 5, 7–9, 15, 20–24, 26–28, 30, 36, 43, 44, 47, 50, 56, 57, 62, 68–71, 73, 83, 84, 86–88, 90–93, 95, 97–99, 102–105, 108, 109, 111, 112, 114–118, 120–122, 124–128, 130, 132, 134, 137, 141–148, 154, 157–159, 161, 162, 164, 165, 181

SCP (Secure Copy Protocol)

Protokoll für sichere Dateiübertragungen über ein zuverlässiges Netzwerk und Standard-Dateiübertragungsprotokoll für die Verwendung mit SSH. 19

Secure Channel

Bezeichnung in der Kryptografie für ein Schema zur sicheren Übertragung über ein unsicheres Medium. 4, 14, 50

SFTP (SSH File Transfer Protocol)

Protokoll für sichere Dateiübertragungen über ein zuverlässiges Netzwerk und Standard-Dateiübertragungsprotokoll für die Verwendung mit SSH-2 [59]. 14, 19, 37, 38, 43, 62, 65, 109–113, 149, 150, 156, 176

SHA (Secure Hash Algorithm)

Hash-Funktion. 7, 8, 13

SHA-1 (Secure Hash Algorithm 1)

Hash-Funktion (Verwendung wird vom NIST nicht mehr empfohlen [20]). 7, 8, 13, 78, *siehe* SHA (Secure Hash Algorithm)

SHA-2 (Secure Hash Algorithm 2)

Hash-Funktion. 7, 8, 13, 26, 28, 29, 36, 42, *siehe* SHA (Secure Hash Algorithm)

SHA-256 (Secure Hash Algorithm 2 mit 256-Bits)

Hash-Funktion. 7, 8, 13, 26, 28, 29, 39, 78, 109, 110, 112, 113, *siehe* SHA-2 (Secure Hash Algorithm 2)

SHA-3 (Secure Hash Algorithm 3)

Hash-Funktion. 42, 162, *siehe* SHA (Secure Hash Algorithm)

SHA-512 (Secure Hash Algorithm 2 mit 512-Bits)

Hash-Funktion. 8, 13, 26, 28, 29, 162, *siehe* SHA-2 (Secure Hash Algorithm 2)

Shell

Software zur Interaktion mit einem Betriebssystem. 12, 24, 25, 37, 39, 66, 105, 107, 108

SSH (Secure Shell)

Protokoll für sicheren Fernzugriff und andere sichere Netzwerkdienste über ein unsicheres Netzwerk [11]. ii, 1–6, 10, 12–19, 21–26, 31–35, 37–41, 43–46, 50, 51, 55, 57, 58, 60–66, 68–70, 72–75, 77, 80–89, 92, 93, 95–99, 102, 104–108, 110–114, 116, 118, 120–122, 124, 126, 127, 129, 134, 137, 140, 141, 143, 145, 147, 149–154, 156–159, 164–166, 176, 177, 181

SSH-2 (auch SSH2)

Secure Shell Protokoll Version 2.0. 4, 13, 166, *siehe* SSH (Secure Shell)

sshd

SSH-Serverdienst von OpenSSH [46]. 3, 54

Streamlined NTRU Prime

Asymmetrischer Verschlüsselungsalgorithmus, der theoretisch resistent gegen Angriffe von Quantencomputern ist [63][64][144]. 26, 50, 98

TCP/IP

Transmission Control Protocol/Internet Protocol
Gruppe von Netzwerkprotokollen. 4, 6, 12

Telnet

Protokoll, das eine entfernte Shell über das Netzwerk ermöglicht (durch SSH ersetzt [171]). 13

UMAC

MAC-Algorithmus mit „Universal Hashing“ [172]. 26

umask

Oktale Maske, mit welcher die Standard-Berechtigungen für neue Verzeichnisse und Dateien eingeschränkt werden können [173]. 32

Unix Socket

Datenkommunikationsendpunkt zum Datenaustausch zwischen Prozessen, kann u. a. über das Dateisystem angesprochen werden. 17, 31, 32, 35, 36, 45, 69, 105, 152

URI (Uniform Resource Identifier)

Zeichenfolge, welche eine logische oder physische Ressource identifiziert. 16, 19

VirtualBox

Virtualisierungssoftware der Firma Oracle. 3, 51, 53, 54, 141, 144

VPN (Virtual Private Network)

Mechanismus zum Aufbau einer sicheren Verbindung zwischen einem Host und einem Netzwerk oder zwei Netzwerken über ein unsicheres Medium. 17, 32

X11 (auch X Window System, X Version 11, X)

Netzwerkprotokoll und Software zur Darstellung von Fenster auf Bitmap-Displays. 12, 24, 32, 105

X25519

ECDH-Funktion mit Operation auf elliptischer Kurve Curve25519. 8, 26

X448

ECDH-Funktion mit Operation auf elliptischer Kurve Curve448. 8

YubiKey

Hardware Security-Token zur Identifizierung und Authentisierung. ii, 55, 82–90, 141–146, 148–151, 157, 159, 162, 177, 181

Zertifikat (auch Public-Key-Zertifikat)

Bindung einer Identität zu einem Public-Key. ii, 8, 18, 20, 22, 26, 30, 36, 43, 44, 50, 55, 71–77, 85–92, 97, 98, 105–107, 130–136, 149–152, 154, 156, 157, 159, 165, 177, 181

Literaturverzeichnis

[1] *OpenBSD Artwork*. URL: https://www.openbsd.org/artwork.html (besucht am 19. 08. 2023).

[2] *OpenSSH Users*. URL: https://www.openssh.com/users.html (besucht am 21. 05. 2023).

[3] *sshd_config — OpenSSH daemon configuration file*. 3. März 2023. URL: https://man.openbsd.org/sshd_config (besucht am 15. 07. 2023).

[4] *OpenSSH Portable Release*. URL: https://www.openssh.com/portable.html (besucht am 15. 06. 2023).

[5] *OpenBSD -stable*. URL: https://www.openbsd.org/stable.html (besucht am 16. 09. 2023).

[6] *Oracle VM VirtualBox*. URL: https://www.virtualbox.org/ (besucht am 07. 07. 2023).

[7] *Debian, Ubuntu, and Raspbian Mirror at ETH Zurich: Debian 5 amd64 ISO*. 22. Jan. 2011. URL: http://debian.ethz.ch/debian-cd/5.0.8/amd64/iso-cd/debian-508-amd64-CD-1.iso (besucht am 13. 09. 2023).

[8] *pfSense - World's Most Trusted Open Source Firewall*. URL: https://www.pfsense.org (besucht am 09. 08. 2023).

[9] Chris M. Lonvick und Sami Lehtinen. *The Secure Shell (SSH) Protocol Assigned Numbers*. RFC 4250. Jan. 2006. DOI: 10.17487/RFC4250. URL: https://www.rfc-editor.org/info/rfc4250.

[10] *OpenSSH Specifications*. URL: https://www.openssh.com/specs.html (besucht am 15. 06. 2023).

[11] Chris M. Lonvick und Tatu Ylonen. *The Secure Shell (SSH) Protocol Architecture*. RFC 4251. Jan. 2006. DOI: 10.17487/RFC4251. URL: https://www.rfc-editor.org/info/rfc4251.

[12] *OpenSSH Project History*. URL: http://www.openssh.com/history.html (besucht am 07. 07. 2023).

[13] *OpenSSH Goals*. URL: https://www.openssh.com/goals.html (besucht am 13. 07. 2023).

[14] Chris M. Lonvick und Tatu Ylonen. *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253. Jan. 2006. DOI: 10.17487/RFC4253. URL: https://www.rfc-editor.org/info/rfc4253.

[15] Chris M. Lonvick und Tatu Ylonen. *The Secure Shell (SSH) Authentication Protocol*. RFC 4252. Jan. 2006. DOI: 10.17487/RFC4252. URL: https://www.rfc-editor.org/info/rfc4252.

[16] Chris M. Lonvick und Tatu Ylonen. *The Secure Shell (SSH) Connection Protocol*. RFC 4254. Jan. 2006. DOI: 10.17487/RFC4254. URL: https://www.rfc-editor.org/info/rfc4254.

[17] National Institute of Standards und Technology (NIST). *Digital Signature Standard (DSS)*. FIPS 186-5. Feb. 2023. DOI: 10.6028/NIST.FIPS.186-5. URL: https://csrc.nist.gov/pubs/fips/186-5/final.

[18] Wesley Griffin und Jakob Schlyter. *Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints*. RFC 4255. Jan. 2006. DOI: 10.17487/RFC4255. URL: https://www.rfc-editor.org/info/rfc4255.

[19] K. Robert Glenn und Stephen Kent. *The NULL Encryption Algorithm and Its Use With IPsec*. RFC 2410. Nov. 1998. DOI: 10.17487/RFC2410. URL: https://www.rfc-editor.org/info/rfc2410.

[20] Elaine Barker und Allen Roginsky. *Transitioning the Use of Cryptographic Algorithms and Key Lengths*. NIST SP 800-131A Rev. 2. März 2019. DOI: 10.6028/NIST.SP.800-131Ar2. URL: https://csrc.nist.gov/pubs/sp/800/131/a/r2/final.

[21] Mark D. Baushke. *Key Exchange (KEX) Method Updates and Recommendations for Secure Shell (SSH)*. RFC 9142. Jan. 2022. DOI: 10.17487/RFC9142. URL: https://www.rfc-editor.org/info/rfc9142.

[22] Chanathip Namprempre, Tadayoshi Kohno und Mihir Bellare. *The Secure Shell (SSH) Transport Layer Encryption Modes*. RFC 4344. Jan. 2006. DOI: 10.17487/RFC4344. URL: https://www.rfc-editor.org/info/rfc4344.

[23] Mark D. Baushke und denis bider. *SHA-2 Data Integrity Verification for the Secure Shell (SSH) Transport Layer Protocol*. RFC 6668. Juli 2012. DOI: 10.17487/RFC6668. URL: https://www.rfc-editor.org/info/rfc6668.

[24] David Carrel und Dan Harkins. *The Internet Key Exchange (IKE)*. RFC 2409. Nov. 1998. DOI: 10.17487/RFC2409. URL: https://www.rfc-editor.org/info/rfc2409.

[25] Loganaden Velvindron und Mark D. Baushke. *Increase the Secure Shell Minimum Recommended Diffie-Hellman Modulus Size to 2048 Bits*. RFC 8270. Dez. 2017. DOI: 10.17487/RFC8270. URL: https://www.rfc-editor.org/info/rfc8270.

[26] Mika Kojo und Tero Kivinen. *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*. RFC 3526. Mai 2003. DOI: 10.17487/RFC3526. URL: https://www.rfc-editor.org/info/rfc3526.

[27] Elaine Barker u. a. *Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography*. NIST SP 800-56A Rev. 3. Apr. 2018. DOI: 10.6028/NIST.SP.800-56Ar3. URL: https://csrc.nist.gov/pubs/sp/800/56/a/r3/final.

[28] Markus Friedl, Niels Provos und William A. Simpson. *Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol*. RFC 4419. März 2006. DOI: 10.17487/RFC4419. URL: https://www.rfc-editor.org/info/rfc4419.

[29] Douglas Stebila und Jonathan Green. *Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer*. RFC 5656. Dez. 2009. DOI: 10.17487/RFC5656. URL: https://www.rfc-editor.org/info/rfc5656.

[30] Bodo Moeller u. a. *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*. RFC 4492. Mai 2006. DOI: 10.17487/RFC4492. URL: https://www.rfc-editor.org/info/rfc4492.

[31] Daniel R. L. Brown. *Recommended Elliptic Curve Domain Parameters*. SEC 2, ver. 2.0. Jan. 2010. URL: https://secg.org/sec2-v2.pdf.

[32] Aris Adamantiadis, Simon Josefsson und Mark D. Baushke. *Secure Shell (SSH) Key Exchange Method Using Curve25519 and Curve448*. RFC 8731. Feb. 2020. DOI: 10.17487/RFC8731. URL: https://www.rfc-editor.org/info/rfc8731.

[33] denis bider. *Use of RSA Keys with SHA-256 and SHA-512 in the Secure Shell (SSH) Protocol*. RFC 8332. März 2018. DOI: 10.17487/RFC8332. URL: https://www.rfc-editor.org/info/rfc8332.

[34] Ben Harris und Loganaden Velvindron. *Ed25519 and Ed448 Public Key Algorithms for the Secure Shell (SSH) Protocol*. RFC 8709. Feb. 2020. DOI: 10.17487/RFC8709. URL: https://www.rfc-editor.org/info/rfc8709.

[35] denis bider. *Extension Negotiation in the Secure Shell (SSH) Protocol*. RFC 8308. März 2018. DOI: 10.17487/RFC8308. URL: https://www.rfc-editor.org/info/rfc8308.

[36] Martin Forssen und Frank Cusack. *Generic Message Exchange Authentication for the Secure Shell Protocol (SSH)*. RFC 4256. Jan. 2006. DOI: 10.17487/RFC4256. URL: https://www.rfc-editor.org/info/rfc4256.

[37] François Yergeau. *UTF-8, a transformation format of ISO 10646*. RFC 3629. Nov. 2003. DOI: 10.17487/RFC3629. URL: https://www.rfc-editor.org/info/rfc3629.

[38] Ondřej Surý. *Use of the SHA-256 Algorithm with RSA, Digital Signature Algorithm (DSA), and Elliptic Curve DSA (ECDSA) in SSHFP Resource Records*. RFC 6594. Apr. 2012. DOI: 10.17487/RFC6594. URL: https://www.rfc-editor.org/info/rfc6594.

[39] S Moonesamy. *Using Ed25519 in SSHFP Resource Records*. RFC 7479. März 2015. DOI: 10.17487/RFC7479. URL: https://www.rfc-editor.org/info/rfc7479.

[40] Phillip Remaker und Joseph Galbraith. *The Secure Shell (SSH) Session Channel Break Extension*. RFC 4335. Jan. 2006. DOI: 10.17487/RFC4335. URL: https://www.rfc-editor.org/info/rfc4335.

[41] Joseph A. Salowey u. a. *Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol*. RFC 4462. Mai 2006. DOI: 10.17487/RFC4462. URL: https://www.rfc-editor.org/info/rfc4462.

[42] Rodney L. Thayer und Joseph Galbraith. *The Secure Shell (SSH) Public Key File Format*. RFC 4716. Nov. 2006. DOI: 10.17487/RFC4716. URL: https://www.rfc-editor.org/info/rfc4716.

[43] Simon Tatham und Darren Tucker. *IUTF8 Terminal Mode in Secure Shell (SSH)*. RFC 8160. Apr. 2017. DOI: 10.17487/RFC8160. URL: https://www.rfc-editor.org/info/rfc8160.

[44] Sami Lehtinen und Tatu Ylonen. *SSH File Transfer Protocol. draft-ietf-secsh-filexfer-02.txt*. Internet-Draft draft-ietf-secsh-filexfer-02. Work in Progress. Internet Engineering Task Force, Okt. 2001. 29 S. URL: https://datatracker.ietf.org/doc/draft-ietf-secsh-filexfer/02/.

[45] Joseph Galbraith und Oskari Saarenmaa. *SSH File Transfer Protocol*. Internet-Draft draft-ietf-secsh-filexfer-extensions-00. Work in Progress. Internet Engineering Task Force, Jan. 2006. 12 S. URL: https://datatracker.ietf.org/doc/draft-ietf-secsh-filexfer-extensions/00/.

[46] *OpenSSH main page*. URL: https://www.openssh.com (besucht am 21. 05. 2023).

[47] *ssh — OpenSSH remote login client*. 21. Juni 2023. URL: https://man.openbsd.org/ssh.1 (besucht am 15. 07. 2023).

[48] *scp — OpenSSH secure file copy*. 16. Dez. 2022. URL: https://man.openbsd.org/scp.1 (besucht am 15. 07. 2023).

[49] *sftp — OpenSSH secure file transfer*. 16. Dez. 2022. URL: https://man.openbsd.org/sftp.1 (besucht am 15. 07. 2023).

[50] *ssh-add — adds private key identities to the OpenSSH authentication agent*. 4. Feb. 2023. URL: https://man.openbsd.org/ssh-add.1 (besucht am 15. 07. 2023).

[51] *ssh-keysign — OpenSSH helper for host-based authentication*. 21. März 2023. URL: https://man.openbsd.org/ssh-keysign.8 (besucht am 15. 07. 2023).

[52] *ssh-keyscan — gather SSH public keys from servers*. 10. Feb. 2023. URL: https://man.openbsd.org/ssh-keyscan.1 (besucht am 15. 07. 2023).

[53] *ssh-keygen — OpenSSH authentication key utility*. 10. Feb. 2023. URL: https://man.openbsd.org/ssh-keygen.1 (besucht am 15. 07. 2023).

[54] *sshd — OpenSSH daemon*. 10. Feb. 2023. URL: https://man.openbsd.org/sshd.8 (besucht am 15. 07. 2023).

[55] *sftp-server — OpenSSH SFTP server subsystem*. 21. Juli 2021. URL: https://man.openbsd.org/sftp-server.8 (besucht am 15. 07. 2023).

[56] *ssh-agent — OpenSSH authentication agent*. 7. Okt. 2022. URL: https://man.openbsd.org/ssh-agent.1 (besucht am 15. 07. 2023).

[57] *ssh_config — OpenSSH client configuration file*. 27. März 2023. URL: https://man.openbsd.org/ssh_config.5 (besucht am 15. 07. 2023).

[58] *OpenSSH for OpenBSD*. URL: https://www.openssh.com/openbsd.html (besucht am 07. 07. 2023).

[59] Joseph Galbraith und Oskari Saarenmaa. *SSH File Transfer Protocol. draft-ietf-secsh-filexfer-13.txt*. Internet-Draft draft-ietf-secsh-filexfer-13. Work in Progress. Internet Engineering Task Force, Juli 2006. 60 S. URL: https://datatracker.ietf.org/doc/draft-ietf-secsh-filexfer/13/.

[60] Michael W. Lucas. *SSH Mastery - Second Edition. OpenSSH, PuTTY, Tunnels and Keys*. 6. Feb. 2018.

[61] Simon Josefsson und Sean Leonard. *Textual Encodings of PKIX, PKCS, and CMS Structures*. RFC 7468. Apr. 2015. DOI: 10.17487/RFC7468. URL: https://www.rfc-editor.org/info/rfc7468.

[62] Sean Turner. *Asymmetric Key Packages*. RFC 5958. Aug. 2010. DOI: 10.17487/RFC5958. URL: https://www.rfc-editor.org/info/rfc5958.

[63] *OpenSSH Release Notes*. URL: https://www.openssh.com/releasenotes.html (besucht am 16. 09. 2023).
[64] Scott Fluhrer. *Quantum Cryptanalysis of NTRU*. Cryptology ePrint Archive, Paper 2015/676. https://eprint.iacr.org/2015/676. Juli 2015. URL: https://eprint.iacr.org/2015/676.
[65] *login.conf — login class capability database*. 31. März 2022. URL: https://man.openbsd.org/login.conf.5 (besucht am 17. 07. 2023).
[66] *Kerberos, AFS and SSH for your Understanding*. 28. Sep. 2004. URL: https://www.zeuthen.desy.de/technisches_seminar/texte/Kerberos-AFS-SSH.pdf (besucht am 17. 07. 2023).
[67] *Kerberos Authentication*. URL: https://www.fortinet.com/resources/cyberglossary/kerberos-authentication (besucht am 17. 07. 2023).
[68] *Scalable and secure access with SSH*. 12. Sep. 2016. URL: https://engineering.fb.com/2016/09/12/security/scalable-and-secure-access-with-ssh/ (besucht am 07. 07. 2023).
[69] *SSH Tunneling: Examples, Command, Server Config*. URL: https://www.ssh.com/academy/ssh/tunneling-example (besucht am 16. 07. 2023).
[70] ISO/IEC. *Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*. Standard ISO/IEC 7498-1:1994. International Organization for Standardization, Juni 1996. 59 S. URL: https://www.iso.org/standard/20269.html.
[71] *syslog, [...] — control system log*. 31. März 2022. URL: https://man.openbsd.org/syslog.3 (besucht am 17. 07. 2023).
[72] *Internet Protocol*. RFC 791. Sep. 1981. DOI: 10.17487/RFC0791. URL: https://www.rfc-editor.org/info/rfc791.
[73] Bob Hinden und Dr. Steve E. Deering. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460. Dez. 1998. DOI: 10.17487/RFC2460. URL: https://www.rfc-editor.org/info/rfc2460.
[74] Fred Baker, Jozef Babiarz und Kwok Ho Chan. *Configuration Guidelines for DiffServ Service Classes*. RFC 4594. Aug. 2006. DOI: 10.17487/RFC4594. URL: https://www.rfc-editor.org/info/rfc4594.
[75] *Service Name and Transport Protocol Port Number Registry*. 29. Juni 2023. URL: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml (besucht am 17. 07. 2023).
[76] IEEE und The Open Group. "IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(TM)) Base Specifications, Issue 7". In: *IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)* (2018), S. 1–3951. DOI: 10.1109/IEEESTD.2018.8277153.
[77] Informatiksicherheit Bund SEC. *IT-Grundschutz in der Bundesverwaltung*. Si001 – IT-Grundschutz in der Bundesverwaltung - Version 5.0. Feb. 2022. URL: https://www.ncsc.admin.ch/dam/ncsc/de/dokumente/dokumentation/vorgaben/sicherheit/si001/Si001-IT-Grundschutz_V5-0-d.pdf.download.pdf/Si001-IT-Grundschutz_V5-0-d.pdf.
[78] FUB ZEO KRYPT. *Empfehlungen zu kryptografischen Verfahren für den Grundschutz*. Jan. 2023.
[79] Bundesamt für Sicherheit in der Informationstechnik. *Kryptographische Verfahren: Empfehlungen und Schlüssellängen*. BSI - Technische Richtlinie, Kryptographische Verfahren: Empfehlungen und Schlüssellängen. Jan. 2023. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf.
[80] Elaine Barker. *Guideline for Using Cryptographic Standards in the Federal Government: Cryptographic Mechanisms*. NIST SP 800-175B Rev. 1. März 2020. DOI: 10.6028/NIST.SP.800-175Br1. URL: https://csrc.nist.gov/Pubs/sp/800/175/b/r1/Final.
[81] *sshd_config - How to Configure the OpenSSH Server?* URL: https://www.ssh.com/academy/ssh/sshd_config (besucht am 23. 07. 2023).
[82] *Eight ways to protect SSH access on your system*. 29. Okt. 2020. URL: https://www.redhat.com/sysadmin/eight-ways-secure-ssh (besucht am 23. 07. 2023).

[83] *CIS Benchmarks*. URL: https://learn.cisecurity.org/benchmarks (besucht am 15. 08. 2023).
[84] Center for Internet Security Inc. (CIS). *CIS Debian Linux 11 Benchmark*. Version 1.0.0. 22. Sep. 2022.
[85] Center for Internet Security Inc. (CIS). *CIS Red Hat Enterprise Linux 9 Benchmark*. Version 1.0.0. 28. Nov. 2022.
[86] Center for Internet Security Inc. (CIS). *CIS Distribution Independent Linux*. Version 2.0.0. 16. Juli 2019.
[87] Daniel J. Barrett, Richard E. Silverman und Robert G. Byrnes. *SSH, The Secure Shell: The Definitive Guide, 2nd Edition*. Mai 2005.
[88] Michael Kofler u. a. *Hacking & Security, 3., aktualisierte und erweiterte Auflage. Das umfassende Handbuch*. Dez. 2022.
[89] *SafeCurves: choosing safe curves for elliptic-curve cryptography*. URL: https://safecurves.cr.yp.to (besucht am 22. 07. 2023).
[90] Johannes Merkle und Manfred Lochter. *Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation*. RFC 5639. März 2010. DOI: 10.17487/RFC5639. URL: https://www.rfc-editor.org/info/rfc5639.
[91] Lily Chen u. a. *Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters*. NIST SP 800-186. Feb. 2023. DOI: 10.6028/NIST.SP.800-186. URL: https://csrc.nist.gov/Pubs/sp/800/186/Final.
[92] *Fail2Ban*. 28. Sep. 2011. URL: https://www.fail2ban.org/wiki/index.php/Fail2Ban (besucht am 23. 07. 2023).
[93] Joint Task Force. *Assessing Security and Privacy Controls in Information Systems and Organizations*. NIST SP 800-53A Rev. 5. Jan. 2022. DOI: 10.6028/NIST.SP.800-53Ar5. URL: https://csrc.nist.gov/pubs/sp/800/53/a/r5/final.
[94] *How To Harden OpenSSH on Ubuntu 20.04*. 8. Nov. 2021. URL: https://www.digitalocean.com/community/tutorials/how-to-harden-openssh-on-ubuntu-20-04 (besucht am 23. 07. 2023).
[95] *Strace Spelunking: Diving Deep into SSH Password Discovery*. 6. Sep. 2023. URL: https://www.sueks.io/strace-spelunking-diving-deep-into-ssh-password-discovery/ (besucht am 20. 09. 2023).
[96] *Bug 3316 - possible bypass of fido 2 devices and ssh-askpass*. URL: https://bugzilla.mindrot.org/show_bug.cgi?id=3316 (besucht am 29. 07. 2023).
[97] *OpenSSH Security*. URL: https://www.openssh.com/security.html (besucht am 29. 07. 2023).
[98] *Common Vulnerability Scoring System v3.1: Specification Document*. URL: https://www.first.org/cvss/v3.1/specification-document (besucht am 29. 07. 2023).
[99] *CVE - Frequently Asked Questions (FAQs)*. URL: https://www.cve.org/ResourcesSupport/FAQs (besucht am 29. 07. 2023).
[100] *CVE List V5*. URL: https://github.com/CVEProject/cvelistV5 (besucht am 29. 07. 2023).
[101] *CVE*. URL: https://www.cve.org (besucht am 29. 07. 2023).
[102] *CVE-2020-5917*. URL: https://www.cve.org/CVERecord?id=CVE-2020-5917 (besucht am 29. 07. 2023).
[103] *CVE-2021-36368*. URL: https://www.cve.org/CVERecord?id=CVE-2021-36368 (besucht am 29. 07. 2023).
[104] *CVE-2023-28531*. URL: https://www.cve.org/CVERecord?id=CVE-2023-28531 (besucht am 29. 07. 2023).
[105] *CVE-2023-38408*. URL: https://www.cve.org/CVERecord?id=CVE-2023-38408 (besucht am 29. 07. 2023).
[106] *CVE-2023-38408 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2023-38408 (besucht am 29. 07. 2023).
[107] *CVE-2023-28531 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2023-28531 (besucht am 29. 07. 2023).

[108] *CVE-2023-25136*. URL: https://www.cve.org/CVERecord?id=CVE-2023-25136 (besucht am 29. 07. 2023).
[109] *CVE-2023-25136 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2023-25136 (besucht am 29. 07. 2023).
[110] *CVE-2021-41617*. URL: https://www.cve.org/CVERecord?id=CVE-2021-41617 (besucht am 29. 07. 2023).
[111] *CVE-2021-41617 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2021-41617 (besucht am 29. 07. 2023).
[112] *CVE-2021-36368*. URL: https://www.cve.org/CVERecord?id=CVE-2021-36368 (besucht am 29. 07. 2023).
[113] *CVE-2021-36368 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2021-36368 (besucht am 29. 07. 2023).
[114] *CVE-2021-28041*. URL: https://www.cve.org/CVERecord?id=CVE-2021-28041 (besucht am 29. 07. 2023).
[115] *CVE-2021-28041 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2021-28041 (besucht am 29. 07. 2023).
[116] *CVE-2020-15778*. URL: https://www.cve.org/CVERecord?id=CVE-2020-15778 (besucht am 29. 07. 2023).
[117] *CVE-2020-15778 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2020-15778 (besucht am 29. 07. 2023).
[118] *CVE-2020-15778*. URL: https://docs.ssh-mitm.at/vulnerabilities/CVE-2020-15778.html (besucht am 29. 07. 2023).
[119] *CVE-2020-14145*. URL: https://www.cve.org/CVERecord?id=CVE-2020-14145 (besucht am 29. 07. 2023).
[120] *CVE-2020-14145 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2020-14145 (besucht am 29. 07. 2023).
[121] *CVE-2020-14145*. URL: https://docs.ssh-mitm.at/vulnerabilities/CVE-2020-14145.html (besucht am 29. 07. 2023).
[122] *CVE-2020-12062*. URL: https://www.cve.org/CVERecord?id=CVE-2020-12062 (besucht am 29. 07. 2023).
[123] *CVE-2020-12062 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2020-12062 (besucht am 29. 07. 2023).
[124] *CVE-2019-16905*. URL: https://www.cve.org/CVERecord?id=CVE-2019-16905 (besucht am 29. 07. 2023).
[125] *CVE-2019-16905 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2019-16905 (besucht am 29. 07. 2023).
[126] *CVE-2019-6111*. URL: https://www.cve.org/CVERecord?id=CVE-2019-6111 (besucht am 29. 07. 2023).
[127] *CVE-2019-6111 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2019-6111 (besucht am 29. 07. 2023).
[128] *CVE-2019-6110*. URL: https://www.cve.org/CVERecord?id=CVE-2019-6110 (besucht am 29. 07. 2023).
[129] *CVE-2019-6110 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2019-6110 (besucht am 29. 07. 2023).
[130] *CVE-2019-6109*. URL: https://www.cve.org/CVERecord?id=CVE-2019-6109 (besucht am 29. 07. 2023).
[131] *CVE-2019-6109 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2019-6109 (besucht am 29. 07. 2023).
[132] *CVE-2018-20685*. URL: https://www.cve.org/CVERecord?id=CVE-2018-20685 (besucht am 29. 07. 2023).

[133] *CVE-2018-20685 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2018-20685 (besucht am 29. 07. 2023).
[134] *CVE-2018-15919*. URL: https://www.cve.org/CVERecord?id=CVE-2018-15919 (besucht am 29. 07. 2023).
[135] *CVE-2018-15919 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2018-15919 (besucht am 29. 07. 2023).
[136] *CVE-2018-15473*. URL: https://www.cve.org/CVERecord?id=CVE-2018-15473 (besucht am 29. 07. 2023).
[137] *CVE-2018-15473 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2018-15473 (besucht am 29. 07. 2023).
[138] *CWE - Common Weakness Enumeration*. URL: https://cwe.mitre.org/ (besucht am 29. 07. 2023).
[139] *OpenBSD CVS - src/usr.bin/ssh/*. URL: https://cvsweb.openbsd.org/cgi-bin/cvsweb/src/usr.bin/ssh/ (besucht am 29. 07. 2023).
[140] *OpenBSD manual page server: syspatch — manage base system binary patches*. 7. Dez. 2020. URL: https://man.openbsd.org/syspatch.8 (besucht am 29. 07. 2023).
[141] *OpenBSD manual page server: afterboot — things to check after the first complete boot*. 15. März 2023. URL: https://man.openbsd.org/afterboot (besucht am 18. 06. 2023).
[142] *OpenBSD FAQ - System Management*. URL: https://www.openbsd.org/faq/faq10.html (besucht am 18. 06. 2023).
[143] *OpenBSD FAQ - Package Management*. URL: https://www.openbsd.org/faq/faq15.html (besucht am 18. 06. 2023).
[144] Gorjan Alagic u. a. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. NIST IR 8413. Sep. 2022. DOI: 10.6028/NIST.IR.8413-upd1. URL: https://csrc.nist.gov/Pubs/ir/8413/upd1/Final.
[145] *Oracle VM VirtualBox User Manual - Configuring Virtual Machines - Motherboard Tab*. URL: https://www.virtualbox.org/manual/ch03.html#settings-motherboard (besucht am 18. 06. 2023).
[146] *OpenBSD 7.3 on VirtualBox 7: Installation as guest OS failed due to I/O APIC enabled*. 6. Mai 2023. URL: https://obsd.solutions/en/blog/2023/05/06/openbsd-73-on-virtualbox-7-installation-as-guest-os-failed-due-to-io-apic-enabled/index.html.
[147] *OpenBSD FAQ - Installation Guide*. URL: https://www.openbsd.org/faq/faq4.html (besucht am 18. 06. 2023).
[148] *OpenBSD FAQ - Networking*. URL: https://www.openbsd.org/faq/faq6.html (besucht am 09. 08. 2023).
[149] *pkg_add — install or update software packages*. 12. Aug. 2022. URL: https://man.openbsd.org/pkg_add (besucht am 09. 08. 2023).
[150] *OpenBSD manual page server: su — substitute user identity*. 22. Dez. 2022. URL: https://man.openbsd.org/su (besucht am 12. 08. 2023).
[151] *OpenBSD manual page server: nsd - Name Server Daemon (NSD) version 4.7.0*. 7. Juni 2023. URL: https://man.openbsd.org/nsd (besucht am 13. 08. 2023).
[152] *Domain names - implementation and specification*. RFC 1035. Nov. 1987. DOI: 10.17487/RFC1035. URL: https://www.rfc-editor.org/info/rfc1035.
[153] *FIDO2 SSH Authentication*. 6. Juli 2023. URL: https://github.com/Yubico/developers.yubico.com/blob/142e9bbf97e61125154556e13dd55794ff6368f3/content/SSH/Securing_SSH_with_FIDO2.adoc (besucht am 14. 08. 2023).
[154] *YubiKey Manager*. URL: https://www.yubico.com/support/download/yubikey-manager/ (besucht am 14. 08. 2023).
[155] *Using PIV for SSH through PKCS #11*. URL: https://developers.yubico.com/PIV/Guides/SSH_with_PIV_and_PKCS11.html (besucht am 14. 08. 2023).
[156] *ssh resident keys and ssh agent*. 29. Juli 2022. URL: https://www.reddit.com/r/yubikey/comments/wbdurr/ssh_resident_keys_and_ssh_agent/ (besucht am 15. 08. 2023).

[157] *OpenBSD Security*. URL: https://www.openbsd.org/security.html (besucht am 29. 07. 2023).
[158] *OpenBSD FAQ - Building the System from Source*. URL: https://www.openbsd.org/faq/faq5.html (besucht am 16. 09. 2023).
[159] Doug Whiting, Russ Housley und Niels Ferguson. *Counter with CBC-MAC (CCM)*. RFC 3610. Sep. 2003. DOI: 10.17487/RFC3610. URL: https://www.rfc-editor.org/info/rfc3610.
[160] Morris Dworkin. *Digital Signature Standard (DSS)*. NIST SP 800-38D. Nov. 2007. DOI: 10.6028/NIST.SP.800-38D. URL: https://csrc.nist.gov/pubs/sp/800/38/d/final.
[161] Joseph A. Salowey, David McGrew und Abhijit Choudhury. *AES Galois Counter Mode (GCM) Cipher Suites for TLS*. RFC 5288. Aug. 2008. DOI: 10.17487/RFC5288. URL: https://www.rfc-editor.org/info/rfc5288.
[162] *Why you should use a BSD style license for your Open Source Project*. 3. Nov. 2021. URL: https://docs.freebsd.org/en/articles/bsdl-gpl/ (besucht am 15. 06. 2023).
[163] Yoav Nir und Adam Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 8439. Juni 2018. DOI: 10.17487/RFC8439. URL: https://www.rfc-editor.org/info/rfc8439.
[164] Simon Josefsson und Ilari Liusvaara. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. RFC 8032. Jan. 2017. DOI: 10.17487/RFC8032. URL: https://www.rfc-editor.org/info/rfc8032.
[165] Jeff Hodges u. a. *Web Authentication: An API for accessing Public Key Credentials Level 2*. Apr. 2021. URL: https://www.w3.org/TR/webauthn-2/.
[166] *LibreSSL*. URL: https://www.libressl.org/ (besucht am 09. 08. 2023).
[167] Paul van Oorschrot Alfred Menezes und Scott Vanstone. *Handbook of Applied Cryptography - Key Establishment Protocols. Key Establishment Protocols*. CRC Press, 1996. URL: https://cacr.uwaterloo.ca/hac/about/chap12.pdf (besucht am 07. 07. 2023).
[168] *pty, ptm — pseudo terminal driver*. 13. Okt. 2022. URL: https://man.openbsd.org/pty.4 (besucht am 17. 07. 2023).
[169] *tty, cua — general terminal interface*. 18. Feb. 2022. URL: https://man.openbsd.org/tty.4 (besucht am 17. 07. 2023).
[170] *rtable, rdomain — routing tables and routing domains*. 9. Juli 2022. URL: https://man.openbsd.org/rdomain.4 (besucht am 19. 07. 2023).
[171] D. Valenčić und V. Mateljan. *Implementation of NETCONF Protocol*. Mai 2019. DOI: 10.23919/MIPRO.2019.8756925. URL: https://ieeexplore.ieee.org/document/8756925.
[172] Ted Krovetz. *UMAC: Message Authentication Code using Universal Hashing*. RFC 4418. März 2006. DOI: 10.17487/RFC4418. URL: https://www.rfc-editor.org/info/rfc4418.
[173] *umask — set file creation mode mask*. 18. Feb. 2022. URL: https://man.openbsd.org/umask (besucht am 19. 07. 2023).

Versionsverzeichnis

Datum	Tätigkeit	Aufwand ca.
21.05.2023	Erstellung Projektskizze Vorbereitung LaTeX Vorlage	8h
07.06.2023	Themenpräsenation Anpassung Projektskizze	1h
15.06.2023	Entwurf Einleitung und Vorgehen	3h
18.06.2023	Test-Installation und Troubleshooting mit OpenBSD-VM	3h
21.06.2023	Kickoff-Meeting mit Experte (Hansjürg Wenger)	1h
07.07.2023	Definition Massnahmen zur Zielerarbeitung Recherche SSH-Protokoll „Core RFCs“ gemäss OpenSSH-Webseite [10]	6h
13.07.2023	Recherche SSH-Protokoll „Core RFCs“ und „Extension RFCs“ gemäss OpenSSH-Webseite [10]	12h
15.07.2023	Recherche SFTP und OpenSSH-Anwendungen `ssh`, `scp`, `sftp`, `ssh-add`, `ssh-keysign`, `ssh-keyscan`, `ssh-keygen`	8h
16.07.2023	Recherche OpenSSH-Anwendung `sshd` und zugehörige Konfigurationsdatei `sshd_config` inkl. Local- und Remote Forwarding bis und mit Parameter `AllowUsers`	5h
17.07.2023	Recherche `sshd`-Konfigurationsdatei `sshd_config` bis und mit Parameter `PubkeyAuthentication`	8h
19.07.2023	Recherche `sshd`-Konfigurationsdatei `sshd_config` und OpenSSH-Anwendungen `sftp-server` und `ssh-agent`	4h
22.07.2023	Recherche Publikationen von Bundesbehörden Ermittlung Minimalstandard	6h
23.07.2023	Recherche weiterer Artikel und Publikationen Ermittlung Minimalstandard	4h
29.07.2023	Vergleich des ermittelten Minimalstandards mit SSH-Server-Standardeinstellungen Recherche Schwachstellen bzw. Verwundbarkeiten Interpretation der Empfehlungen	7h
09.08.2023	50%-Meeting mit Experte (Hansjürg Wenger) Aufbau Laborumgebung Erstellung Beispielausgaben für OpenSSH-Programme	9h
12.08.2023	Prüfen OpenSSH Release Notes [63] und Entscheid mit Version vom 19.07.2023 fortzufahren Einrichtung von OpenSSH in der Laborumgebung inklusive Erstellen eines Private-Keys, der SSH-Server-Grundkonfiguration und der Konfiguration für die Anwendungsfälle „Kommandozeilenzugriff“, „Dateiübertragungen“ und „Jumphost“	10h

Datum	Tätigkeit	Aufwand ca.
13.08.2023	Einrichtung des Authentisierungs-Agenten `ssh-agent` inklusive Agent-Forwarding Implementation mit Zertifikaten, CA, SSHFP-DNS-Records und DNS-Überprüfung auf dem SSH-Server	7h
14.08.2023	FIDO2-Authentisierung mit YubiKey inklusive Troubleshooting mit zugehörigem Zertifikat auf YubiKey und Authentisierungs-Agenten mit YubiKey	4h
15.08.2023	Weiteres Troubleshooting mit zugehörigem Zertifikat auf YubiKey und Authentisierungs-Agenten mit YubiKey Verfassen der Übersicht und Arbeitsflüsse	4h
19.08.2023	Suche und Einbindung Titelbild	1h
26.08.2023	Ermittlung und Auflistung der Testfälle	2h
10.09.2023	Ausarbeitung der Testfälle und Verifikation des Aufbaus in den Kategorien „Allgemein“, „Kommandozeilenzugriff“ und „Dateiübertragungen“	8h
13.09.2023	Ausarbeitung der Testfälle und Verifikation des Aufbaus in den Kategorien „Public-Key-Authentisierung“, „Jumphost“, „Authentisierungs-Agent“ und „SSHFP-DNS-Records“	8h
15.09.2023	90%-Meeting mit Experte (Hansjürg Wenger) Ausarbeitung der Testfälle und Verifikation des Aufbaus in den Kategorien „Zertifikate“ und „FIDO2-Authentisierung mit YubiKey“	7h
16.09.2023	Verfassen von Fazit, Rückblick und Ausblick Erstellen des OpenSSH „Cheat Sheets“ Einbinden der SSH-Server-Konfigurationen in den Anhang	6h
20.09.2023	Erweiterung des OpenSSH „Cheat Sheets“ Abstract und Review	10h
23.09.2023	Präsentation erstellen	6h
24.09.2023	Review und Abgabe	4h
	Total Aufwand	162h

Tabelle 6.1.: Versionsverzeichnis

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die hier vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Sämtliche Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, habe ich als solche kenntlich gemacht.

Hiermit stimme ich zu, dass die vorliegende Arbeit in elektronischer Form mit entsprechender Software überprüft wird.

24. September 2023

Mauro Guadagnini

A. OpenSSH Cheat Sheet

Dieses „Cheat Sheet“ wurde separates Dokument zur Abgabe an der Präsentation produziert.

Client-Parameter

SSH-Verbindung zu Server „srv.example.com“ unter **Port 2222** als Benutzer „user“
SSH-Standardport ist 22
`ssh -p 2222 user@srv.example.com`
Betroffene Server-Optionen:
`Listen 2222`,
`AddressFamily` für IPv4/IPv6,
`ListenAddress` kann auch Port beinhalten

Verbindungsaufbau über einen **Jumphost** „jumphost“
mit Benutzer „jumpuser“
auf Zielhost „target“ mit Benutzer „user“
`ssh -J jumpuser@jumphost user@target`

Betroffene Server-Optionen auf Jumphost:
`DisableForwarding no`, `AllowTcpForwarding yes`, `PermitOpen target:22`, `MaxSessions 0`

Öffnen einer **Local Forwarding** Verbindung, um Anfragen auf den Client-Port 80 über den Server „srv“ nach „www“ auf Port 80 weiterzuleiten
`ssh -L 80:www:80 srv`

Betroffene Server-Optionen auf Jumphost:
`DisableForwarding no`, `AllowTcpForwarding yes`, `PermitOpen target:22`, `MaxSessions 0`

Öffnen einer **Remote Forwarding** Verbindung, um Anfragen auf den Server-Port 8080 über den Server „srv“ nach „localhost“ auf Port 80 weiterzuleiten
`ssh -R 8080:localhost:80 srv`

Betroffene Server-Optionen auf Jumphost:
`DisableForwarding no`, `AllowTcpForwarding yes`, `PermitOpen target:22`, `MaxSessions 0`

Prüfen eines SSHFP-DNS-Records mit dem zugehörigen Server-Fingerprint beim Verbindungsaufbau zu Server „srv“
als Benutzer „user“
`ssh -o "VerifyHostKeyDNS yes" user@srv`
Achtung: Verbindung wird bei unzulässigem SSHFP-Record nicht blockiert
SSHFP-Records mittels `ssh-keygen -r hostname` ausgelesen und via DNS publiziert

Server-Konfiguration `/etc/ssh/sshd_config`

Einrichten, dass **Public-Key- und Passwort-Authentisierung zusammen** erfüllt sein müssen
`AuthenticationMethods publickey,password`

Einrichten der **Dateiübertragung** für Benutzer „file“, welcher **nur innerhalb dem Pfad** „/data/sftp“ operieren darf

```
Match User file
    ForceCommand internal-sftp
    ChrootDirectory /data/sftp
```

`Match`-Blöcke am Ende der Konfigurationsdatei anfügen
Pfad muss Benutzer „root“ gehören, Unterordner darf „file“ gehören („file“ kann somit nichts direkt im Pfad schreiben)

Nur Benutzer der Gruppe „sshaccess“ den SSH-Zugriff erlauben
`AllowGroups sshaccess`
Hierbei geht es um den Zielbenutzer auf dem Server

Ausführung des Befehls „echo hello“ für Benutzer mit Kommandozeilenzugriff **forcieren**
`ForceCommand echo hello`
SSH-Sitzung wird nach Ausführung des Befehls geschlossen
Alternativ kann z.B. ein Skript mit einer Auswahl an Befehlen angegeben werden

Authentisierungs-Agent

Agent in der aktuellen Shell-Sitzung **starten**
`eval $(ssh-agent -s)`
Eventuell könnte dieser bereits gestartet sein, prüfen mit z.B. `pgrep -l ssh-agent`

Schlüssel „~/.ssh/id_ed25519“ **zum Agent hinzufügen mit der Bedingung**, dass dieser nur für die Verbindungen „alice@srv1“ und bei aktivem Agent-Forwarding von „srv1“ nach „bob@srv2“ verwendet werden darf

```
ssh-add -h 'alice@srv1' \
 -h 'srv1>bob@srv2' ~/.ssh/id_ed25519
```

Die Server-Namen müssen hierzu bereits in der Datei `~/.ssh/known_hosts` vorhanden sein
Betroffene Server-Optionen:
`AllowAgentForwarding yes`
Agent-Forwarding mit SSH-Client-Option `-A` aktivieren

Abbildung A.1.: OpenSSH Cheat Sheet Vorderseite

Es folgt eine Ausgabe der **SSH-Server-Konfigurationsdatei**, wie sie auf den Servern dieser Arbeit implementiert wurde, mit sämtlichen angewandten Variationen (farblich markiert). Die Algorithmen-Wahl wurde aus Platzgründen im Cheat Sheet entfernt, stattdessen wird die OpenSSH-Standardauswahl genommen. Folgendes ist zudem zu bemerken:

- Beim ausschliesslichen Einsatz von YubiKeys mit PIN-Abfrage (`PubkeyAuthOptions verify-required` ist bereits hinterlegt) könnte die Passwort-Authentisierung in der Option `AuthenticationMethods` entfernt werden
- Die Option `AllowGroups` hat hinterlegt, dass nur Benutzer der Gruppe „sshaccess“ (hier „cmd“, „file“, „jump“ und „agent“) Zugriff erhalten
- Konfigurierte Zertifikate und Schlüssel sind entsprechend zu erstellen
- Public-Keys und/oder erlaubte Principals (bei Zertifikats-Authentisierung) sind bei den Zielbenutzern in der zugehörigen Datei (`.ssh/authorized_keys` für Public-Keys, `.ssh/authorized_principals` für Principals) zu hinterlegen
- Für Dateiübertragungen mit Benutzer „jump“ ist ein entsprechender Pfad zu erstellen und zu wählen

```
# Listen ----------------------------
Port                    22
AddressFamily           any
ListenAddress           0.0.0.0
ListenAddress           ::
# Private keys of server
HostKey /etc/ssh/ssh_host_rsa_key
HostCertificate /etc/ssh/
    ssh_host_rsa_key-cert.pub
    # RSA host certificate
HostKey /etc/ssh/ssh_host_ecdsa_key
HostCertificate /etc/ssh/
    ssh_host_ecdsa_key-cert.pub
    # ECDSA host certificate
HostKey
    /etc/ssh/ssh_host_ed25519_key
HostCertificate /etc/ssh/
    ssh_host_ed25519_key-cert.pub
    # Ed25519 host certificate

# Algorithms ------------------------
# Ignore to trust OpenSSH selection
#CASignatureAlgorithms
#Ciphers
#HostKeyAlgorithms
#KexAlgorithms
#MACs
#PubkeyAcceptedAlgorithms
RequiredRSASize         3072

# Ciphers and keying ----------------
RekeyLimit          default none
    # Rekey after ciphers default
    # amount, no timebased rekeying

# Logging ---------------------------
SyslogFacility          AUTH
LogLevel                INFO

# Authentication --------------------
LoginGraceTime          1m
StrictModes             yes
AuthenticationMethods
    publickey,password
    # Force PIN when using
    # FIDO auth algo
    # (i.e. ecdsa-sk or ed25519-sk)
    Only allow pubkey +
    # pwauth combined
PubkeyAuthentication    yes
AuthorizedKeysFile
    .ssh/authorized_keys
    # change to "none" to enforce
    # certificate authentication
PubkeyAuthOptions   verify-required
    # Force PIN when using
    # FIDO auth algo
    # (i.e. ecdsa-sk or ed25519-sk)
HostbasedAuthentication no
PasswordAuthentication  yes
PermitEmptyPasswords    no
KbdInteractiveAuthentication no
# Certificate Authentication -------
TrustedUserCAKeys   /etc/ssh/ca.pub
    # Trusted CA
AuthorizedPrincipalsFile
    .ssh/authorized_principals
    # defined principals

# User / Group Filter ---------------
AllowGroups             sshaccess
PermitRootLogin         no

# Forwarding / Tunnel ---------------
DisableForwarding       yes
AllowAgentForwarding    no
AllowStreamLocalForwarding  no
AllowTcpForwarding      no
PermitListen            none
PermitOpen              none
GatewayPorts            no
X11Forwarding           no
PermitTunnel            no

# Other settings --------------------
PermitTTY           no
    # Don't give user a terminal
ForceCommand            exit
    # exit session
PrintMotd               no
PrintLastLog            no
TCPKeepAlive            yes
PermitUserEnvironment   no
PermitUserRC            no
Compression             no
UseDNS                  no
    # activate if using DNS,
    #not using DNS atm
PidFile             /var/run/sshd.pid
MaxStartups             10:30:100
ChrootDirectory         none
VersionAddendum         none
Banner                  none

# sftp subsystem            sftp
Subsystem                   /usr/libexec/sftp-server

# Use cases -------------------------
# --- commandline -------------------
# insert to configure commandline
# access for user "cmd"
# User cmd ----------------------
Match User cmd
    PermitTTY               yes
    ForceCommand            none

# --- filetransfer ------------------
# insert to configure filetransfer
# access over sftp for user "file"
# in a defined directory
# User file ---------------------
Match User file
    ForceCommand        internal-sftp
    ChrootDirectory         /data/sftp

# --- jumphost ----------------------
# insert to configure jumphost
# access for user "jump"
# and to open a TCP forwarding
# sessions to defined destinations
# User jump ---------------------
Match User jump
    DisableForwarding       no
    AllowTcpForwarding      yes
    PermitOpen 192.168.2.1:22
    MaxSessions             0

# --- agent forwarding --------------
# insert to enable agent-forwarding
# and configure commandline access
# for user "agent"
# User agent --------------------
Match User agent
    AllowAgentForwarding yes
    PermitTTY               yes
    ForceCommand            none
```

Abbildung A.2.: OpenSSH Cheat Sheet Rückseite

B. SSH-Server-Konfigurationsdatei

B.1. Server „s1“ und „s2“ /etc/ssh/sshd_config

Es folgt eine Ausgabe der SSH-Server-Konfigurationsdatei, wie sie auf den Servern `s1` und `s2` dieser Arbeit implementiert wurde, mit sämtlichen, im Vergleich zur Grundkonfiguration aus Kapitel 4.2.2 angewandten Variationen (farblich markiert).

Folgendes ist zusätzlich zu bemerken:

- Beim ausschliesslichen Einsatz von YubiKeys mit PIN-Abfrage [86] könnte die Passwort-Authentisierung in der Option `AuthenticationMethods` entfernt werden
- Mittels `AllowGroups` ist hinterlegt, dass nur Benutzer der Gruppe `sshaccess` Zugriff erhalten Die Benutzer `cmd` , `file` , `jump` und `agent` gehören dieser Gruppe auf dem Server an
- Konfigurierte Zertifikate und Schlüssel sind entsprechend zu erstellen
- Entsprechende Public-Keys und/oder erlaubte Principals (bei Zertifikats-Authentisierung) sind bei den Zielbenutzern in der zugehörigen Datei [87] zu hinterlegen
- Für Dateiübertragungen mit Benutzer `jump` ist ein entsprechender Pfad zu erstellen und zu wählen (siehe Kapitel 4.2.6)

```
# Listen ----------------------------------------------------------------
Port                            22
AddressFamily                   any
ListenAddress                   0.0.0.0
ListenAddress                   ::

# Private keys of server ------------------------------------------------
# Generated RSA server host key is 3072 bit

HostKey                         /etc/ssh/ssh_host_rsa_key
                                # RSA host certificate (see chapter 4.2.8.6)
HostCertificate                 /etc/ssh/ssh_host_rsa_key-cert.pub

HostKey                         /etc/ssh/ssh_host_ecdsa_key
                                # ECDSA host certificate (see chapter 4.2.8.6)
HostCertificate                 /etc/ssh/ssh_host_ecdsa_key-cert.pub

HostKey                         /etc/ssh/ssh\_host_ed25519_key
                                # Ed25519 host certificate (see chapter 4.2.8.6)
HostCertificate                 /etc/ssh/ssh_host_ed25519_key-cert.pub
```

[86] Die Option `PubkeyAuthOptions` mit Wert `verify-required` ist bereits hinterlegt
[87] `.ssh/authorized_keys` für Public-Keys, `.ssh/authorized_principals` für Principals

```
# Algorithms ------------------------------------------------------------
# Used to fulfill the minimalstandard, ignore to trust selection of OpenSSH

CASignatureAlgorithms          ssh-ed25519,ecdsa-sha2-nistp256,ecdsa-sha2-
   nistp384,ecdsa-sha2-nistp521,sk-ssh-ed25519@openssh.com,sk-ecdsa-sha2-
   nistp256@openssh.com,rsa-sha2-512,rsa-sha2-256

# with AEAD, not just aes-cbc or aes-ctr
Ciphers                        aes128-gcm@openssh.com,aes256-gcm@openssh.com

HostKeyAlgorithms              ssh-ed25519-cert-v01@openssh.com,ecdsa-sha2-
   nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,
   ecdsa-sha2-nistp521-cert-v01@openssh.com,sk-ssh-ed25519-cert-v01@openssh.
   com,sk-ecdsa-sha2-nistp256-cert-v01@openssh.com,rsa-sha2-512-cert-
   v01@openssh.com,rsa-sha2-256-cert-v01@openssh.com,ssh-ed25519,ecdsa-sha2-
   nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,sk-ssh-ed25519@openssh.
   com,sk-ecdsa-sha2-nistp256@openssh.com,rsa-sha2-512,rsa-sha2-256

# "ssh -Q kex" shows no dh-group15 for DHKE
# but group18 with bigger size
KexAlgorithms                  diffie-hellman-group16-sha512,diffie-hellman-
   group18-sha512,curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-
   nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521

# Use HMAC
MACs                           hmac-sha2-256-etm@openssh.com,hmac-sha2-512-
   etm@openssh.com,hmac-sha2-256,hmac-sha2-512

RequiredRSASize                3072

PubkeyAcceptedAlgorithms       ssh-ed25519-cert-v01@openssh.com,ecdsa-sha2-
   nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,
   ecdsa-sha2-nistp521-cert-v01@openssh.com,sk-ssh-ed25519-cert-v01@openssh.
   com,sk-ecdsa-sha2-nistp256-cert-v01@openssh.com,rsa-sha2-512-cert-
   v01@openssh.com,rsa-sha2-256-cert-v01@openssh.com,ssh-ed25519,ecdsa-sha2-
   nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,sk-ssh-ed25519@openssh.
   com,sk-ecdsa-sha2-nistp256@openssh.com,rsa-sha2-512,rsa-sha2-256

# Ciphers and keying ----------------------------------------------------
# Rekey after ciphers default amount,
# no time based rekeying
RekeyLimit                     default none

# Logging  --------------------------------------------------------------
SyslogFacility                 AUTH
LogLevel                       INFO
```

```
# Authentication  ------------------------------------------------------------
LoginGraceTime                1m
StrictModes                   yes

# Only allow pubkey and pw auth combined
AuthenticationMethods         publickey,password

PubkeyAuthentication          yes

                              # change to "none" to enforce
                              # certificate authentication (see chapter 4.2.8.5)
#AuthorizedKeysFile           none
AuthorizedKeysFile            .ssh/authorized_keys

# Force PIN when using FIDO auth algo
# (i.e. ecdsa-sk or ed25519-sk)
PubkeyAuthOptions             verify-required

HostbasedAuthentication       no

PasswordAuthentication        yes
PermitEmptyPasswords          no

KbdInteractiveAuthentication  no

# insert to configure certificate authentication with
# trusted CA and defined principals (see chapter 4.2.8.2)
# Certificate Authentication -------------------------------------------------
TrustedUserCAKeys             /etc/ssh/ca.pub
AuthorizedPrincipalsFile      .ssh/authorized_principals

# User / Group Filter --------------------------------------------------------
AllowGroups                   sshaccess
PermitRootLogin               no

# Forwarding / Tunnel --------------------------------------------------------
DisableForwarding             yes
AllowAgentForwarding          no
AllowStreamLocalForwarding    no
AllowTcpForwarding            no
PermitListen                  none
PermitOpen                    none
GatewayPorts                  no
X11Forwarding                 no
PermitTunnel                  no
```

```
# Other settings ------------------------------------------------------------
# Don't give user a terminal and exit session
PermitTTY                       no
ForceCommand                    exit

PrintMotd                       no
PrintLastLog                    no
TCPKeepAlive                    yes
PermitUserEnvironment           no
PermitUserRC                    no

Compression                     no

UseDNS                          no # activate if using DNS, not using DNS atm

PidFile                         /var/run/sshd.pid
MaxStartups                     10:30:100

ChrootDirectory                 none
VersionAddendum                 none
Banner                          none

# sftp subsystem
Subsystem                       sftp      /usr/libexec/sftp-server

# insert to configure commandline access for user "cmd" (see chapter 4.2.3)
# User cmd ------------------------------------------------------------------
Match User cmd
    PermitTTY                   yes
    ForceCommand                none

# insert to configure filetransfer access over sftp for user "file"
# in a defined directory (see chapter 4.2.5)
# User file -----------------------------------------------------------------
Match User file
    ForceCommand                internal-sftp
    ChrootDirectory             /data/sftp

# insert to configure jumphost access for user "jump"
# and to open a TCP forwarding session to the defined destination (see chapter 4.2.6)
# User jump -----------------------------------------------------------------
Match User jump
    DisableForwarding           no
    AllowTcpForwarding          yes
    PermitOpen                  192.168.2.1:22
    MaxSessions                 0

# insert to enable agent-forwarding and configure commandline access for user "agent"
# (see chapter 4.2.7.1)
# User agent ----------------------------------------------------------------
Match User agent
    AllowAgentForwarding        yes
    PermitTTY                   yes
    ForceCommand                none
```

Quelltext B.1: SSH-Server-Konfigurationsdatei `/etc/ssh/sshd_config` mit Variationen auf Server `s1` und `s2`

B.2. Server „s3“ (Debian 5) /etc/ssh/sshd_config

Hierbei handelt es sich um die Standardkonfiguration, welche nach dem Installieren von Debian 5 und dem OpenSSH-Server unter `/etc/ssh/sshd_config` vorzufinden ist. Kommentierte Optionen weisen den zugehörigen Standardwert auf.

```
# Package generated configuration file
# See the sshd(8) manpage for details

# What ports, IPs and protocols we listen for
Port 22
# Use these options to restrict which interfaces/protocols sshd will bind to
#ListenAddress ::
#ListenAddress 0.0.0.0
Protocol 2
# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
#Privilege Separation is turned on for security
UsePrivilegeSeparation yes

# Lifetime and size of ephemeral version 1 server key
KeyRegenerationInterval 3600
ServerKeyBits 768

# Logging
SyslogFacility AUTH
LogLevel INFO

# Authentication:
LoginGraceTime 120
PermitRootLogin yes
StrictModes yes

RSAAuthentication yes
PubkeyAuthentication yes
#AuthorizedKeysFile     %h/.ssh/authorized_keys

# Don't read the user's ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
# For this to work you will also need host keys in /etc/ssh_known_hosts
RhostsRSAAuthentication no
# similar for protocol version 2
HostbasedAuthentication no
# Uncomment if you don't trust ~/.ssh/known_hosts for RhostsRSAAuthentication
#IgnoreUserKnownHosts yes

# To enable empty passwords, change to yes (NOT RECOMMENDED)
PermitEmptyPasswords no

# Change to yes to enable challenge-response passwords (beware issues with
# some PAM modules and threads)
ChallengeResponseAuthentication no

# Change to no to disable tunnelled clear text passwords
#PasswordAuthentication yes
```

```
# Kerberos options
#KerberosAuthentication no
#KerberosGetAFSToken no
#KerberosOrLocalPasswd yes
#KerberosTicketCleanup yes

# GSSAPI options
#GSSAPIAuthentication no
#GSSAPICleanupCredentials yes

X11Forwarding yes
X11DisplayOffset 10
PrintMotd no
PrintLastLog yes
TCPKeepAlive yes
#UseLogin no

#MaxStartups 10:30:60
#Banner /etc/issue.net

# Allow client to pass locale environment variables
AcceptEnv LANG LC_*

Subsystem sftp /usr/lib/openssh/sftp-server

UsePAM yes
```

Quelltext B.2: Standard OpenSSH-Server-Konfiguration `/etc/ssh/sshd_config` auf VM mit Debian 5 (Server `s3`)