HSLU Hochschule Luzern

Verteilte Systeme und Komponenten

Sicherheit in verteilten Systemen

Martin Bättig

Letzte Aktualisierung: 15. Dezember 2022

FH Zentralschweiz

Inhalt

- Übersicht
- Transportlayer-Security (TLS)
- Sessions
- Authentifizierung und Autorisierung

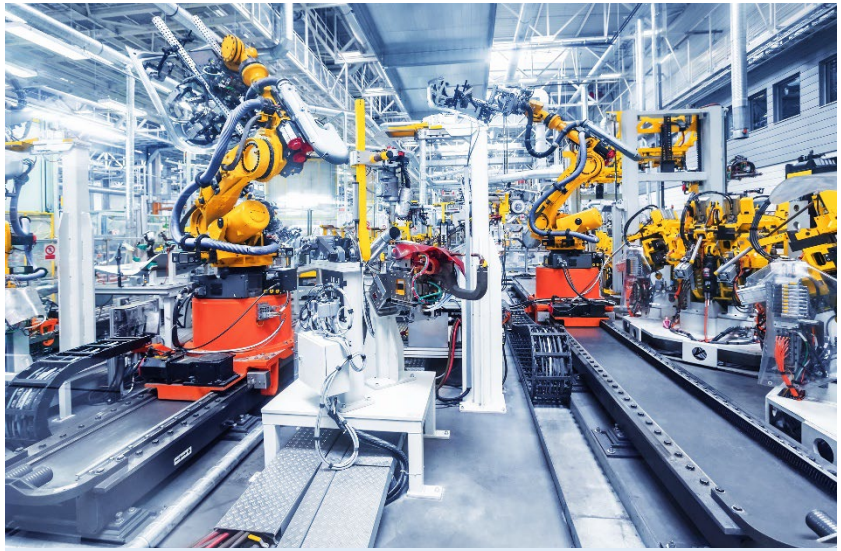
Lernziele

- Sie kennen die notwendigen Massnahmen für eine sichere Kommunikation in verteilten Systemen.
- Sie kennen die Grundlagen des TLS-Protokolls und wie eine TLS-Verbindung auf Socketebene erstellt wird.
- Sie wissen, wie die Erstellung von Zertifikaten in den Grundzügen funktioniert.
- Sie kennen das Prinzip einer Session und können diese in Relation zu einer TCP-Connection setzen.
- Sie kennen den Ablauf einer Authentifizierung mittels Passwort und wie Passwörter mittels Java sicher gespeichert werden können.
- Sie wissen, was eine Autorisierung ist und wie eine einfache Autorisierung mittels Java realisiert werden kann.

Sicherheit in verteilten Systemen

Cyber-Security vs. Betriebssicherheit

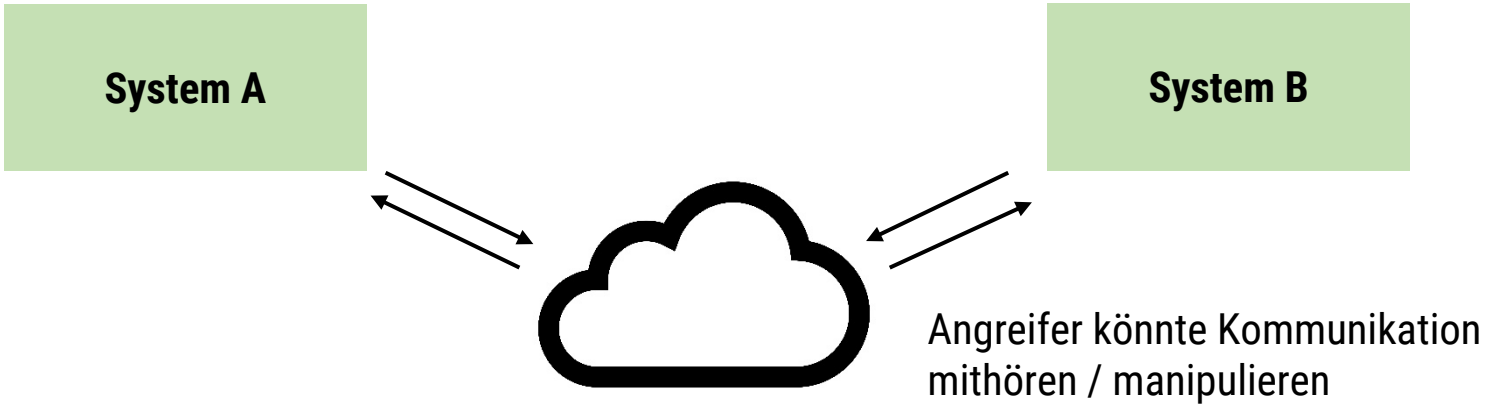
Cyber-Security:
Schutz kritischer Systeme und sensibler Informationen vor digitalen Angriffen.



Betriebssicherheit:
Fehlfunktionen treten nicht auf oder verursachen keine kritischen Schäden an Mensch und Maschine.

Fokus: Sichere Kommunikation zwischen verteilten Systemen

Situation: A und B sollen sicher kommunizieren. Daten gehen über Drittsysteme, eine lange Verbindung oder werden drahtlos übertragen:



Sichere Kommunikation:

1. A muss sicherstellen, dass B das korrekte System ist.
2. B muss sicherstellen, dass A ein berechtigtes System ist.
3. Kein Drittsystem C soll die Kommunikation mithören.
4. Kein Drittsystem C soll die Kommunikation manipulieren.

Dies während der gesamten Dauer der Kommunikation (Sitzung).

Cyber-Security - Massnahmen

- **Zugriffschutz:** Nur berechtige Benutzer und Systeme können auf unserer System regulär zugreifen.
  -> Authentifizierung und Autorisierung.
- **Manipulationssicherheit:** Gesendete Daten können nicht manipuliert werden.
  -> Signaturen.
- **Abhörsicherheit:** Keine geschützte Informationen gelangen nach aussen.
  -> Verschlüsselung.
- **Nachvollziehbarkeit:** Wissen darüber wie und von wem das System verwendet wurde.
  -> Logs / Audit-Trails.

Welche Informationen verbergen?

Tatsache, dass Kommunikation stattfindet, lässt sich nur schwer verbergen (Achtung: Seitenkanäle).

Beispiel: Visible Light Communication.
Kommunikation durch Variation der Frequenz.
Sieht aus wie Licht: Kommunikation für menschliches Auge zunächst verborgen.
Mittels Technik aber erkennbar.

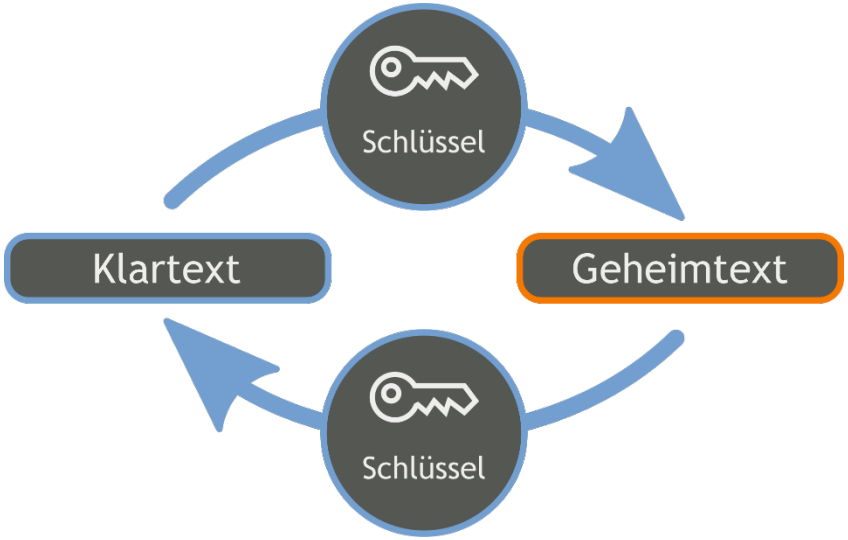
Quelle: Disney Research

Pragmatisches Vorgehen:

- Wenn es einfach ist: Kommunikationsinhalt verbergen (verschlüsseln).
- Wenn es kompliziert wird, gut überlegen, was Sinn ergibt.
- Datenverschlüsselung ist Basis für sichere Datenübertragung.
  - ⇨ Unterscheidung: symmetrische und asymmetrische Verschlüsselung.

Symmetrische Verschlüsselung

Gleicher Schlüssel zum Ver- und Entschlüsseln verwendet:



- Effizienter als asymmetrische Verschlüsselung.
- I.d.R. eingesetzt zum Verschlüsseln von Datenströmen.

Asymmetrische Verschlüsselung

Erzeugung eines Schlüsselpaars:

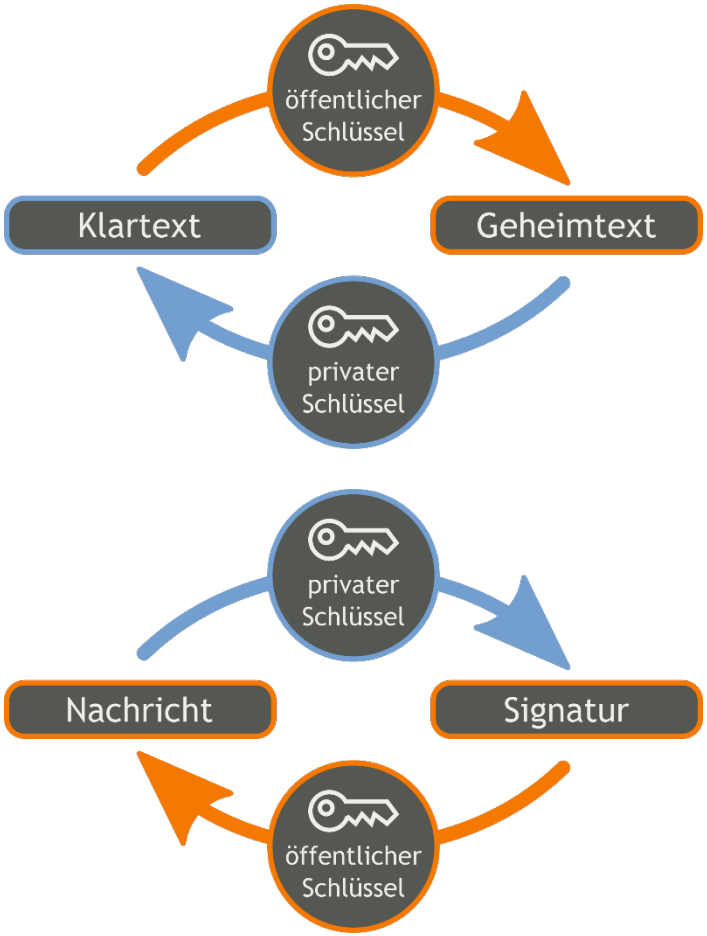
- Privater Schlüssel (nur Erzeuger A bekannt).
- Öffentlicher Schlüssel (allen bekannt, z.B. B).

Einsatz zum Verschlüsseln:

- B verschlüsselt mittels öffentlichem Schlüssel.
- Nur A kann mittels privatem Schlüssel entschlüsseln.

Einsatz um Signieren und Zertifizieren:

- A verschlüsselt Hash einer Information I mittels privatem Schlüssel => Signatur.
- B entschlüsselt Signatur mit öffentlichem Schlüssel. Entspricht diese dem Hashwert von I hat A die Signatur erstellt, da nur A den privaten Schlüssel kennt.

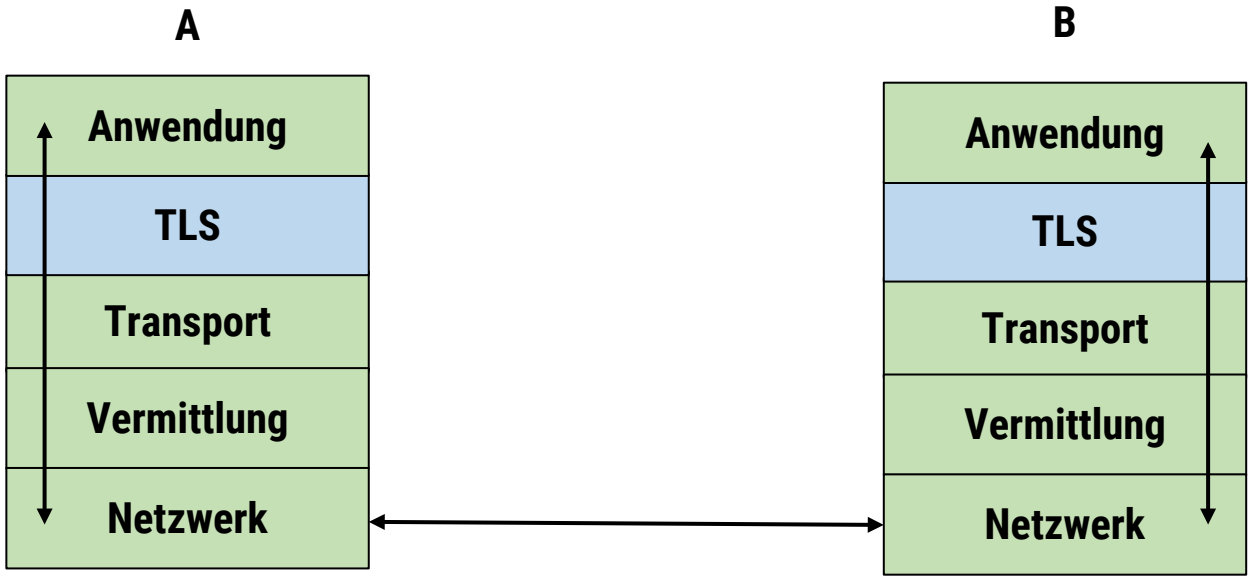


Transportlayer Security (TLS)

Grundlagen

- Transparente Verschlüsselung der Anwendungskommunikation.
- Von Konzept her eine Transportschicht (z.B. TCP).
- Implementiert als Protokoll in der Anwendungsschicht.

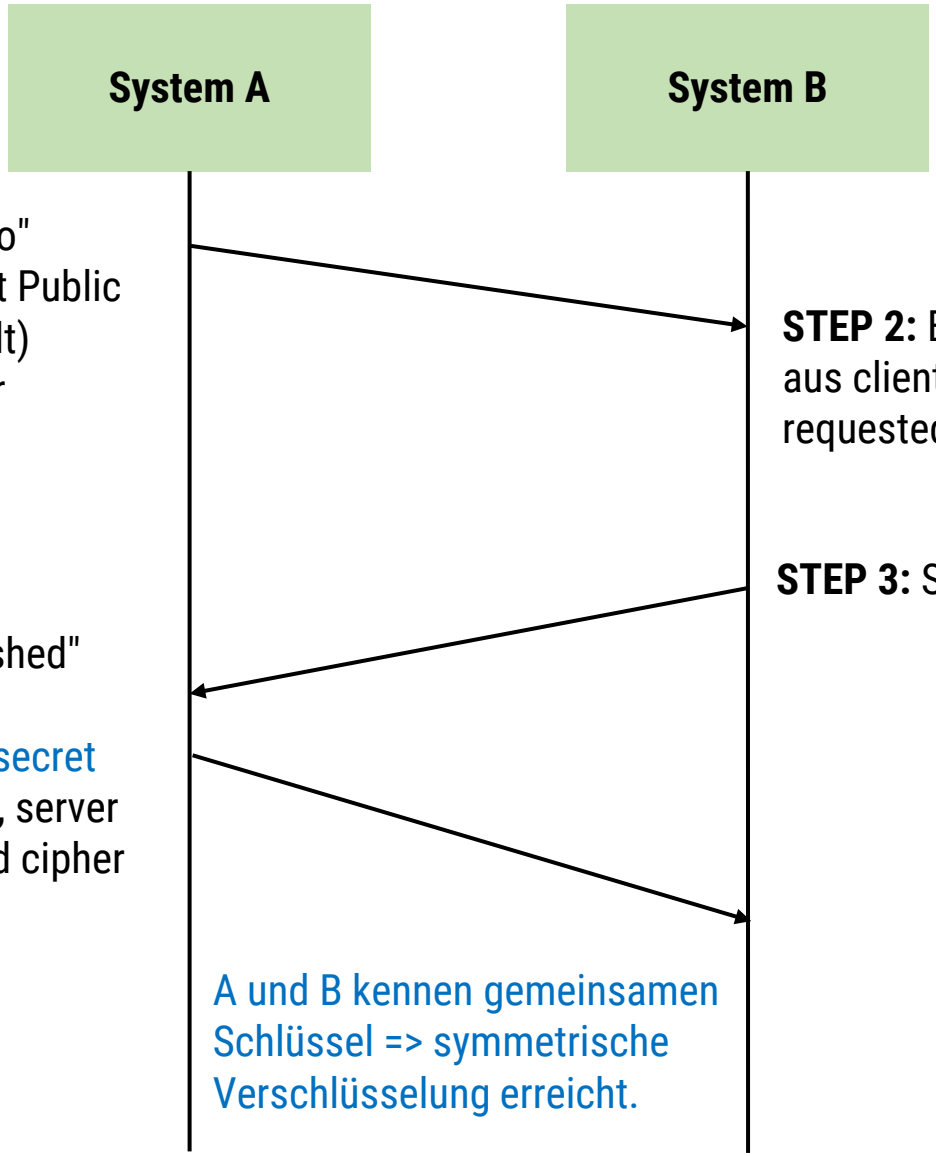
Verortung von TLS innerhalb der Internetschichtenmodells:



Versionen des TLS-Protokolls

Version	Einführung	Info
SSL 1.0	1994	Nicht mehr in Gebrauch, unsicher
SSL 2.0	1995	Nicht mehr in Gebrauch, unsicher
SSL 3.0	1996	Nicht mehr in Gebrauch, unsicher
TLS 1.0	1999	Nicht mehr in Gebrauch, unsicher
TLS 1.1	2006	Nicht mehr in Gebrauch, unsicher
TLS 1.2	2008	In Gebrauch, noch sicher
TLS 1.3	2018	Aktuelle Version, wenn möglich verwenden (RFC 8446)

Verbindungsaufbau (TLS 1.3 Handshake)

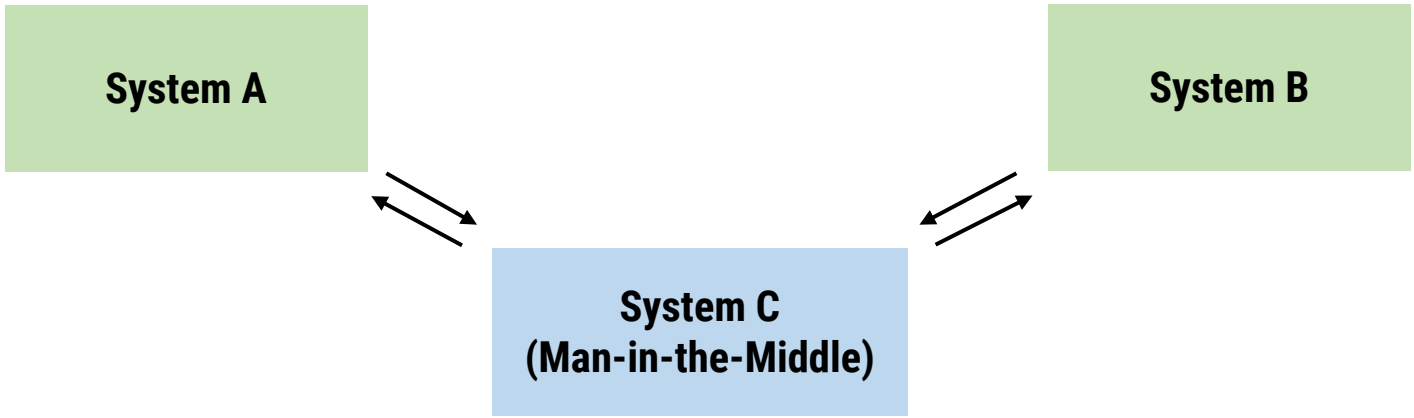


TLS gekoppelt mit Zertifikatsprüfung

- TLS verschlüsselt in Kombination mit Authentifizierung der Gegenstelle.
- Authentifizierung mittels X.509 Zertifikaten.
- Notwendig zur Verhinderung von "Man-in-the-Middle"-Attacken.

Annahme es gäbe keine Authentifizierung:

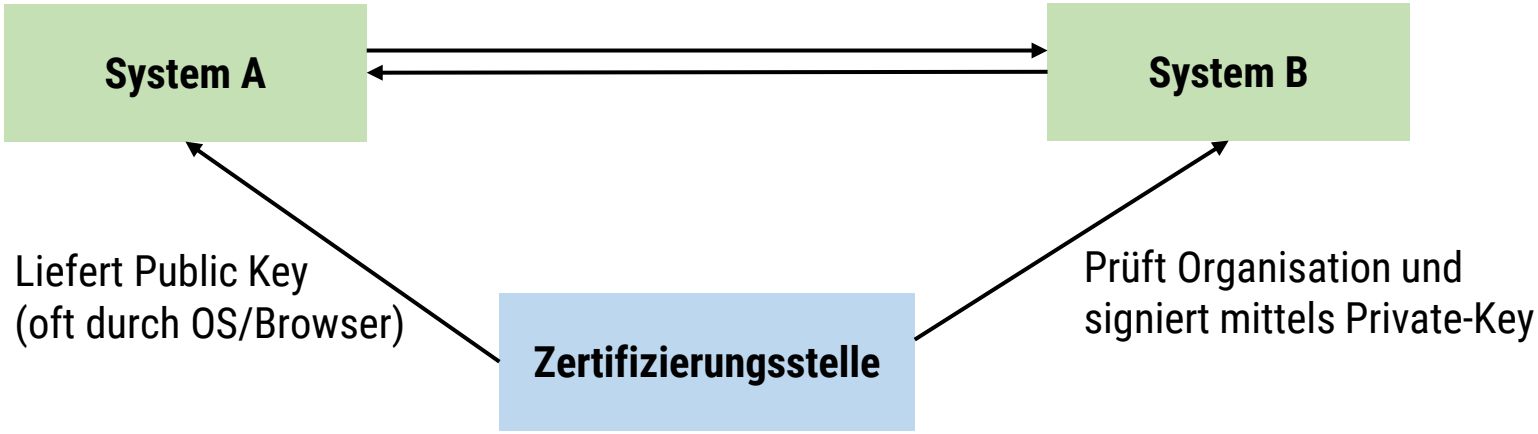
- A stellt Verbindung mit B her und wird via C geroutet:



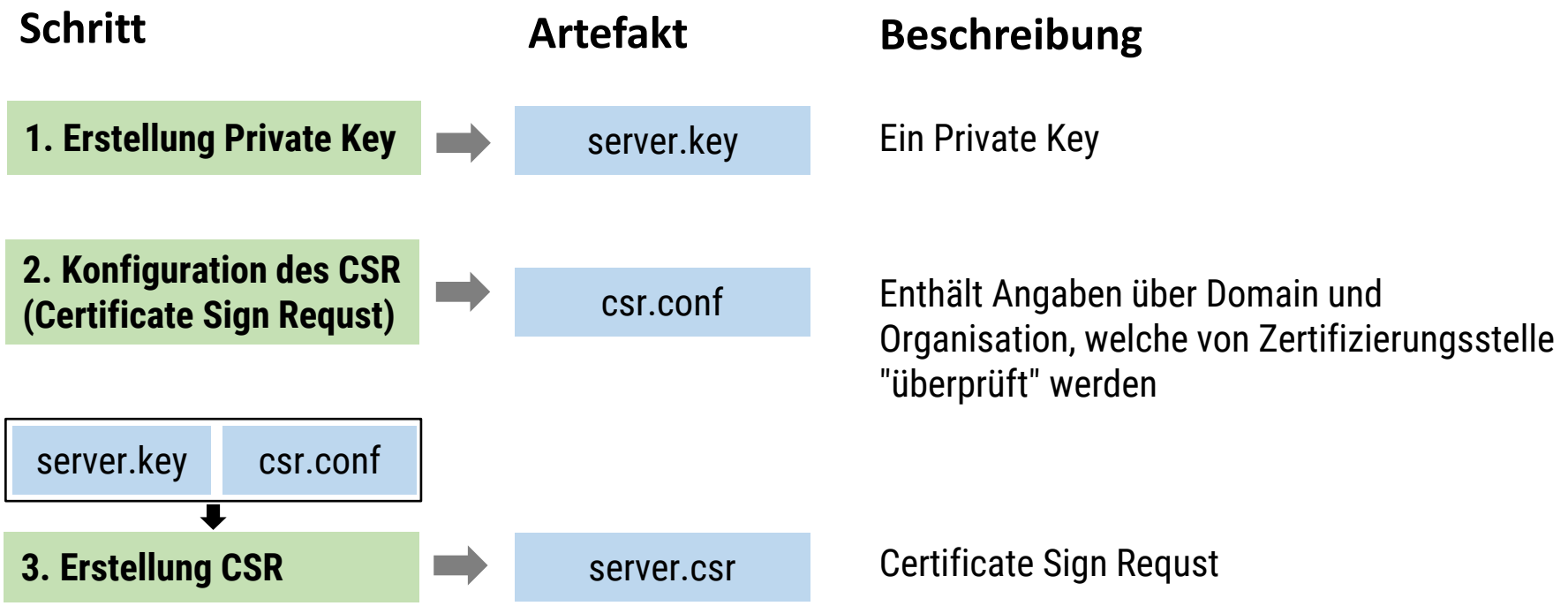
- C könnte beim Aufbau der Verbindung A und B eigene Schlüssel senden und Klartextkommunikation lesen.
  (d.h.: C gibt sich gegenüber A als B aus und gegenüber B als A).

X.509 Zertifikate

- binden eine Identität mittels digitaler Signatur zu öffentlichem Schlüssel.
- sind zertifiziert durch Zertifizierungsstelle oder selbstzertifiziert.
- bedingen Vertrauen in Zertifizierungsstelle (wurde Identität geprüft?).
- unterteilt in zwei Kategorien:
  - **Zertifizierungsstelle (CA):** Kann weitere Zertifikate erteilen. Mehrere Hierarchiestufen. Vertrauen in oberste Hierarchie (Root-CA) benötigt.
  - **Endstelle:** Identifizierte eine Entität (Domain, Person, Organisation, etc.)



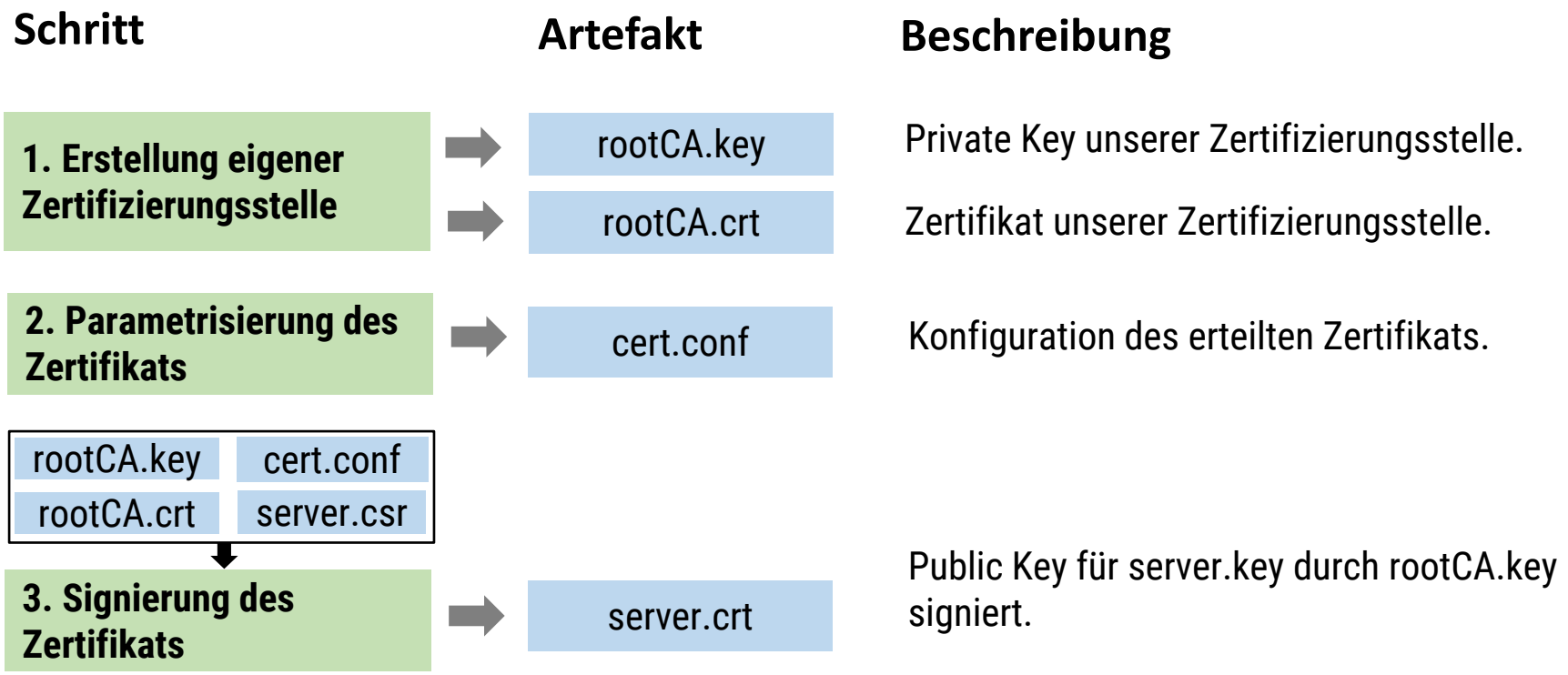
Erstellung eines Certificate Sign Requst



Ausstellung Zertifikat durch Zertifizierungsstelle

Schritt	Artefakt	Beschreibung
server.csr		
1. Senden des CSR und weiteren Informationen an Zertifizierungsstelle	server.crt	Public Key für server.key durch Zertifizierungsstelle signiert.

Zertifikatsausstellung (Selbstzertifiziert)



Aufbau einer sicheren Verbindung mit Java (Keystore)

– Java erwartet Zertifikate und Schlüssel in einem Keystore:

```
## Erstelle PKCS12-Bundle mit Serverzertifikat, Private-Key and, rootCA-Zertifikat)
openssl pkcs12 -export -in server.crt -inkey server.key -chain -CAfile rootCA.crt -name localhost -out server.p12

## Erstelle Keystore der alle Element des PKCS12-Bundle enthält
keytool -importkeystore -deststorepass myServerPass -destkeystore server.jks -srckeystore server.p12 -srcstoretype PKCS12

## create a java keystore that contains the certificate of our own CA
keytool -import -v -trustcacerts -alias server-alias -file rootCA.crt -keystore cacerts.jks -keypass myCaCertsPass -storepass myCaCertsPass
```

Aufbau einer sicheren Verbindung mit Java (Client)

```
private static final String HOST = ...;

public static void main(final String[] args) {

    SSLSocketFactory factory = (SSLSocketFactory)
                               SSLSocketFactory.getDefault();
```

Schritt 1: Erstellung einer SSLSocketFactory

```
    try (SSLSocket socket = (SSLSocket) factory.createSocket(HOST, 1234)) {
```

Schritt 2: Erstellung eines SSLSocket

```
        socket.setEnabledProtocols(new String[] {"TLSv1.3"});
        socket.setEnabledCipherSuites(new String[] {"TLS_AES_128_GCM_SHA256"});
```

Schritt 3: Setzen unterstützter Versionen und Algorithmen

```
        // ab hier: Verwenden wie regulären TCP –Socket
        // ...
    }
}
```

Server: Analog mittels SSLServerSocketFactory vorgehen

Beispiel: Starten von Client und Server mit TLS

Server: Keystore (Name ist beliebig, hier `server.jks`) hier mit Private Key und Zertifikat (immer):

```
java -Djavax.net.ssl.keyStore=server.jks \
     -Djavax.net.ssl.keyStorePassword=myServerPass \
     myServerClassName
```

Client: Keystore (Name ist beliebig, hier `cacerts.jks`) mit Zertifizierungsstelle (nur bei selbsterstellen Zertifikaten):

```
java -Djavax.net.ssl.trustStore=cacerts.jks \
     -Djavax.net.ssl.trustStorePassword=myCaCertsPass \
     myClientClassName
```

Debug-Modus: Falls SSL-Modus nicht funktioniert:

```
-Djavax.net.debug=ssl
```

TLS bei gRPC und ZeroMQ

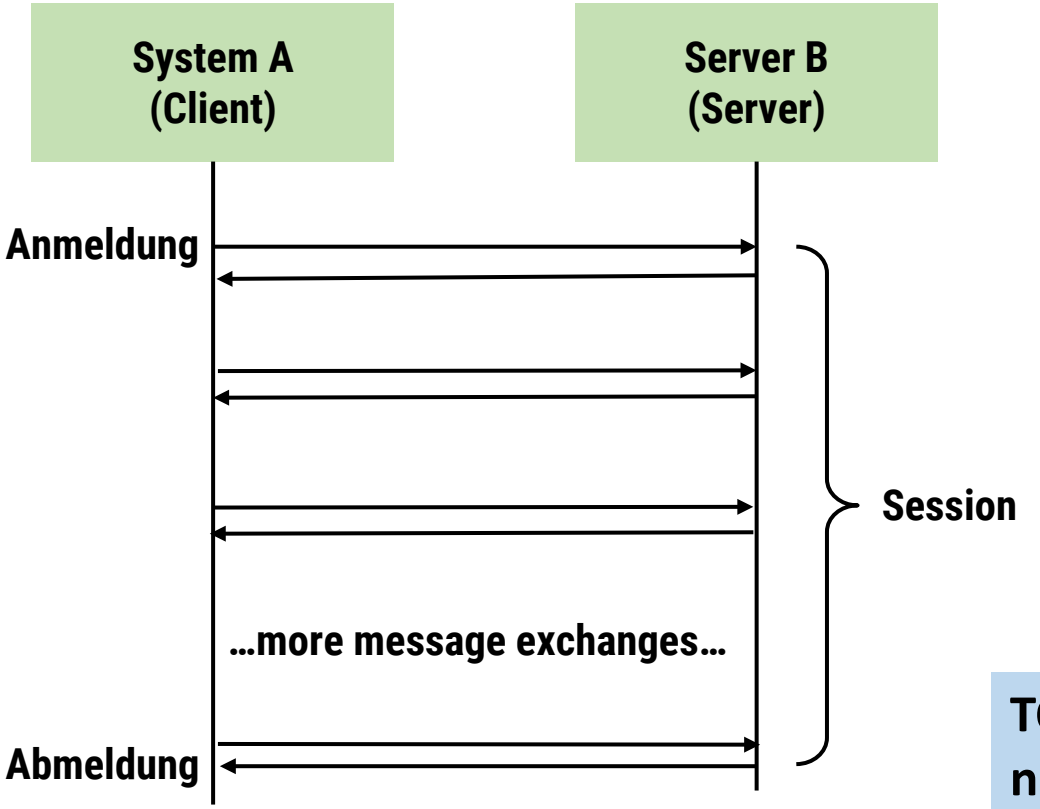
- **gRPC:** Unterstützt TLS per Default
  Anleitung: https://grpc.io/docs/guides/auth/
- **ZeroMQ: E**igenes Transportverschlüsselungsprotokoll basierend auf **CurveCP (http://curvecp.org).**
  - Alternativ: Kommunikation separat mittels Werkzeugen via stunnel (https://www.stunnel.org/config_unix.html) verschlüsseln.

Sessions

Session (Sitzung)

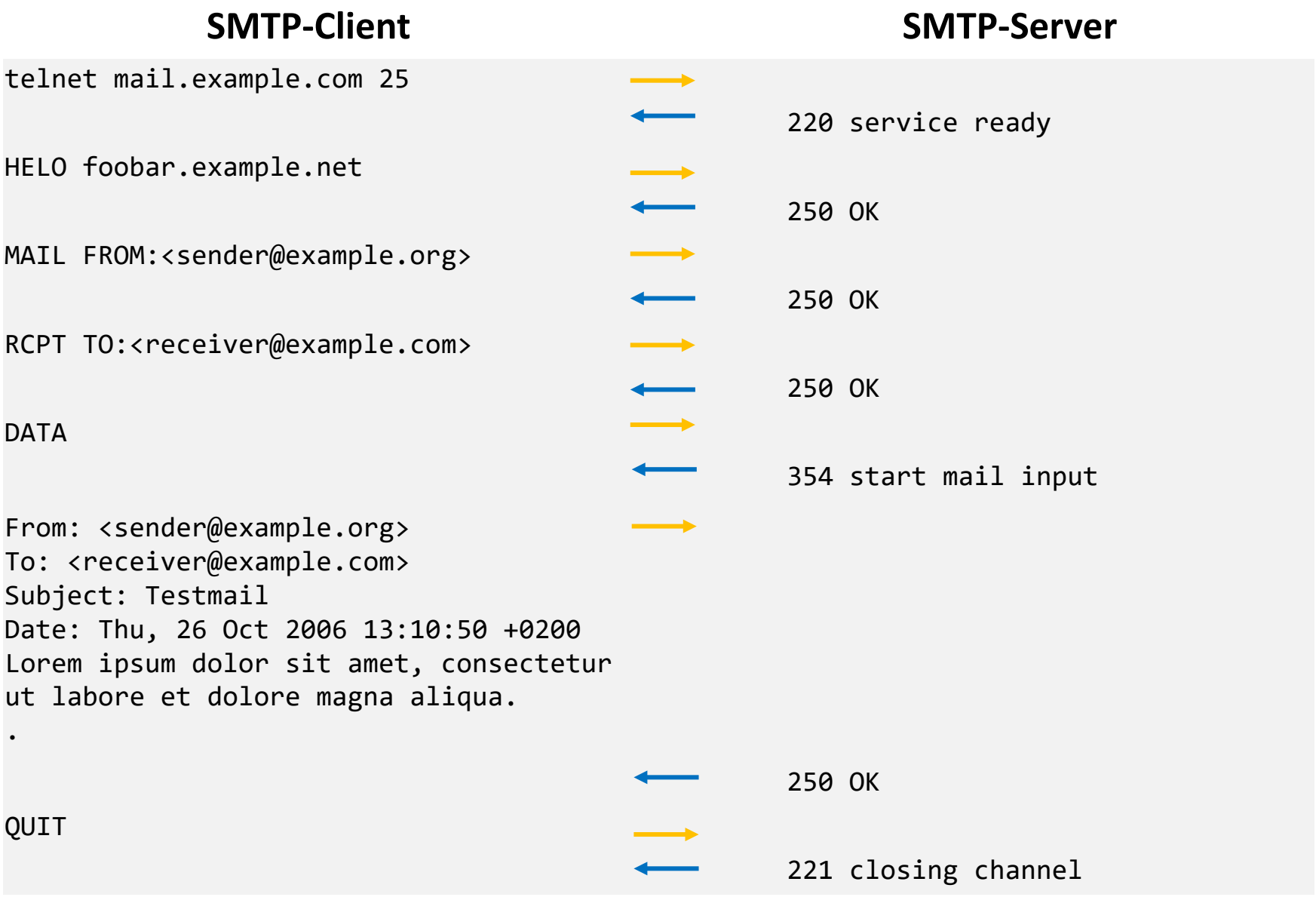
Definition:

- Zeitlich beschränkte Zweiwege-Kommunikation (Anmelden bis Abmelden).
- Typischerweise zwischen Client und Server (Request-Response).



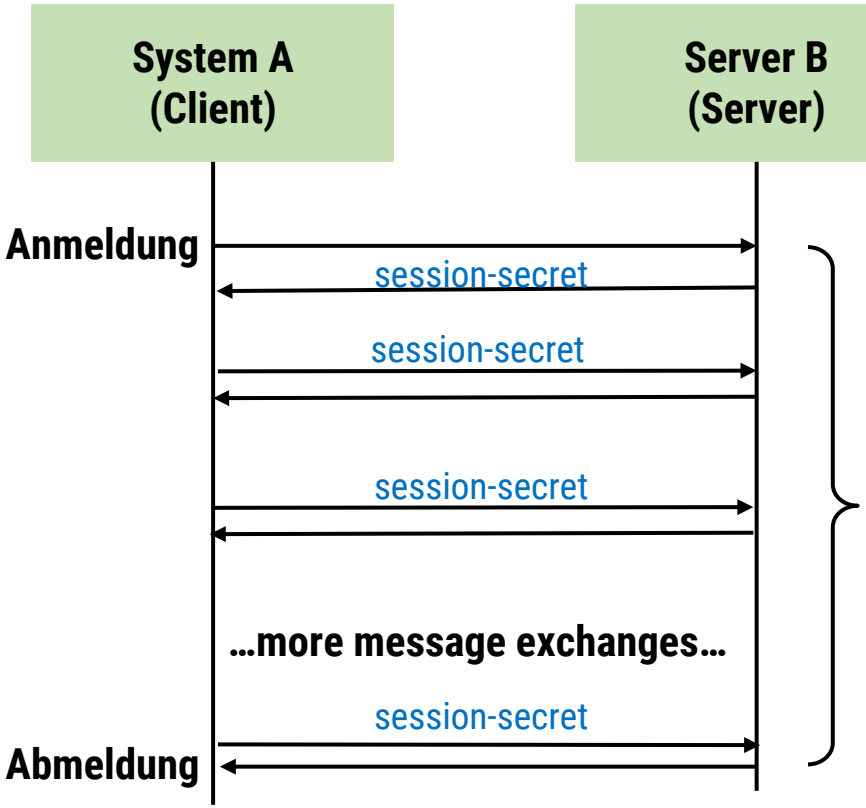
TCP-Verbindung als Session nutzen?

Beispiel: SMTP-Protokoll (Wiederholung)



Session-Secret: Entkopplung der Session von TCP-Connection

– Session-Secret nur dem Server und dem einen Client bekannt.



1. Server erstellt session-secret bei Anmeldung.

2. Client sendet session-secret mit jeder Message.

3. Client und Server löschen session-secret.

Varianten von Session-Secrets

Zufällige Zahl ohne Informationsgehalt:

- Zufallsgenerator muss kryptographisch sicher sein.
- Typische Grössenordnung: 16 bytes.
- Server speichert Informationen, welche zur Session gehören (z.B. Hauptspeicher / Datei / Datenbank /Key-Value Stores (Hazelcast/Redis/etc.).
- **Beispiel:** Session-Cookie in Webapplikationen.

Verschlüsselte statische Informationen (AccessToken):

- Public/Private-Key Verfahren:
  - Kryptographisch (Signatur) gegenüber Veränderung gesichert.
- **Beispiel:** JSONWebToken
  - dient als AccessToken (enthält z.B. Verknüpfung mit Benutzerkonto).
  - Besteht auf Header/Payload/Signature

Beispiel: Berechnung eines Session-Secrets (Zufallszahl)

- Für Java ist SecureRandom eine geeignete Implementation:
- Unter Linux bezieht SecureRandom per Default seine Zahlen von /dev/random (blockiert, falls nicht genügend Entropie).

```
private byte[] generateSessionKey() {
    SecureRandom secureRandom = new SecureRandom();
    byte[] sessionKey = new byte[16];
    secureRandom.nextBytes(sessionKey); // may block
    return sessionKey;
}
```

Beispiel: JSONWebToken

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

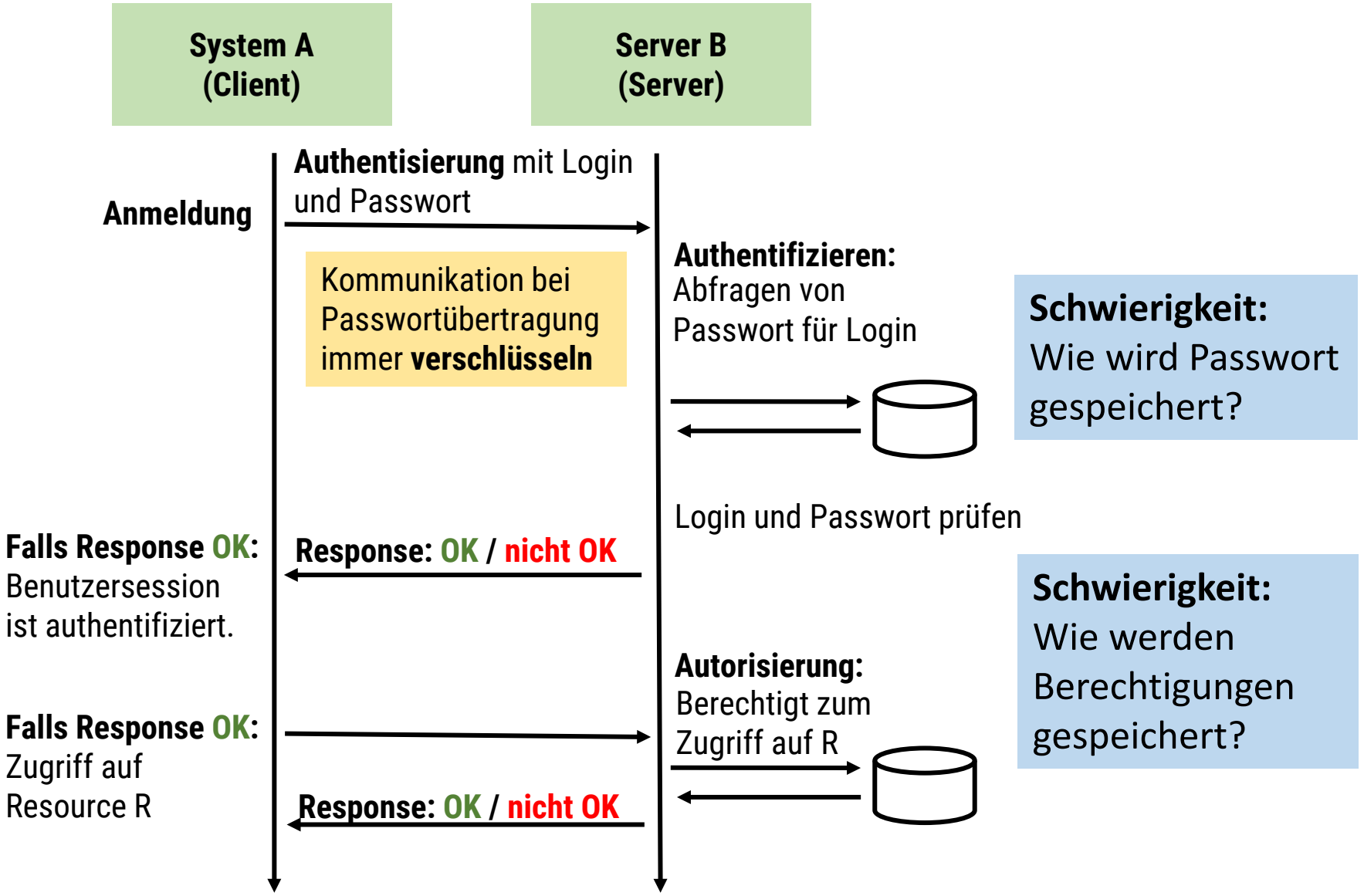
– Quelle: jwt.io

Authentifizierung und Autorisierung

Terminologie

- **Authentisieren:** Eine Partei P (Benutzer/in oder System) weisst sich gegenüber einem System S mittels einem Geheimnis aus:
  - I.d.R. mittels Kenntnis einer Information, einem Zugang, oder einem Besitz, welcher nur Partei P hat (Passwort, AccessToken, Smartcard, Empfang einer Email, etc.).
- **Authentifizieren:** System S prüft, ob Partei P diejenige ist, welcher sie vorgibt zu sein, in dem dieses Geheimnis überprüft wird.
- **Autorisierung:** Überprüfen, ob eine authentifizierte Partei berechtigt ist, auf eine Ressource zu zugreifen.

Authentifizierung einer Session (am Beispiel mit Passwort)



Authentifizierung einer Session (forts.)

- Authentifiziert wird i.d.R. eine Session, welche für eine Partei P steht.
- Nach erfolgreicher Authentifizierung wird eine Session mit dem Konto der Partei P verknüpft.

```
void login(Message message) {
    String account = message.getAccount();
    String password = message.getPassword();
    PasswordRecord record = PasswordManager.getAccount(account);

    if (record.matches(password)) {

        session.setAccount(account);

    }
}
```

Überprüfung des Passworts

Verknüpfung der Benutzersession

- Wesentlicher Teil der Login-Vorgangs ist die Passwortprüfung.

Passwortprüfung

Passwörter als Hashwert gespeichert:

login	hash
anna	$2y$10$bsD5raIzGlwENSF1JY3Wre/JFp9qIlxKldMuuuD3cnk...
joe	$2y$10$ASC9x1HQpq9ruZx/1w7IMebkg4SuQFTubDKcc.ROVvX...
jack	$2y$10$ByfdpPBHYrkhftMPrgQFKOZNJKyt8nzPpDkiTp8HB1m...
alice	$2y$10$km5WnWbOf9zGqSD9.vqDlezRr9R0spSC1v3u/8VJp.G...

- In jedes System wird früher oder später eingebrochen!
- Klartext Passwörter:
  Gefahr für andere Systeme.
- Speichen als vom Passwort abgeleiteter Wert, ein sogenannter Hash.
- Kommt ein Angreifer in Besitz eines Hashwerts kennt er das Passwort nicht.

Rad nicht neu erfinden:

- Sichere Technologie stammt aus den 1970ern.
- Vorgefertigte und geprüfte Funktionen und Libraries verwenden:
  Bei Sicherheitslücken wird man informiert und steht nicht alleine da.

Exkurs: Hashfunktionen

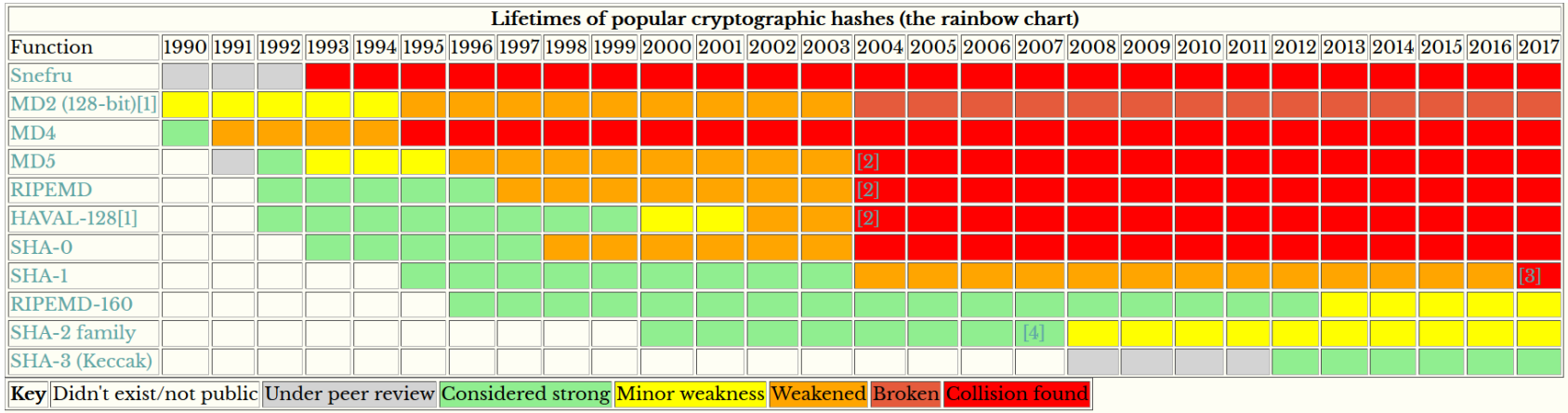
– **Einweg-Funktion:** Aus dem Funktionsresultat lässt sich der Input (das Passwort) nur mit viel Rechenaufwand rekonstruieren:

`=> f(M1s!ecurePW) = Wre/JFp9qIlxKldMuuuD3cnktDfWR` **(billig)**

`=>` f_{Rev}`(Wre/JFp9qIlxKldMuuuD3cnktDfWR) = M1s!ecurePW` **(sehr teuer)**

Geeignete Hashfunktionen

- Keine generische Hash-Funktionen verwenden.
- Diese sind auf Geschwindigkeit optimiert mit beschränkter Lebensdauer:



Quelle: http://valerieaurora.org/hash.html (*2017 Valerie Aurora, licensed CC BY-SA)*

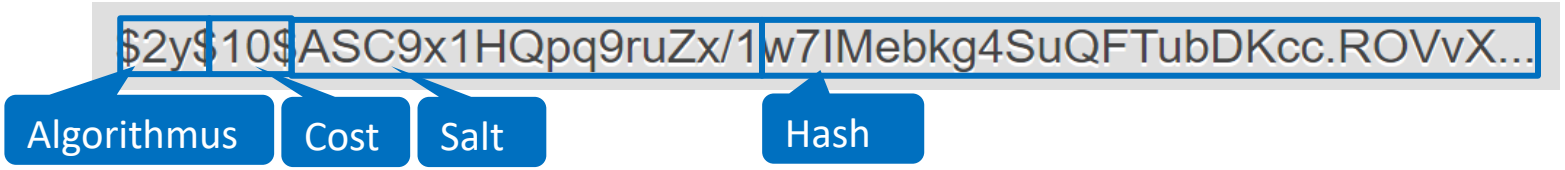
Besser: Passwort-Hashfunktionen verwenden: PBKDF2, Bcrypt, Scrypt.

- Diese sind langsamer in der Berechnung, dies ist aber ein Vorteil.
- PBKDF2 in Kombination mit HMAC-SHA256 empfohlen von NIST [1].

[1] https://pages.nist.gov/800-63-3/sp800-63b.html#memsecretver

Hashfunktionen: Weitere Massnahmen

Beispiel eines Password-Records (hier von PHP):



- **Cost:** Anpassen der Kosten der Hashwertberechnung an aktuelle Hardware.
  - Kosten für ein Login sollte in etwa 100ms sein (immer noch schnell genug für ein Benutzer, aber teuer für Angriffe.
  - Muss jeweils an aktuelle Hardware angepasst werden.
- **Salt:** Verhindert Brute-Force- ("alle Kombinationen durchprobieren") oder Wörterbuch-Attacken.
  - Pro Passwort zufällige Zahl, welche zusammen mit Password «gehasht» wird.
- **Pepper:** (Nicht abgebildet) Zusätzlich alle Passwörter mit weiterer (für alle Passwörter identische) Zahl Z hashen. Diese Zahl Z separat von der Passwort-Db speichern.

Beispiel: Generierung von Salt und Hash in Java

```java
private byte[] generateSalt() {
    SecureRandom random = new SecureRandom();
    byte[] salt = new byte[16];
    random.nextBytes(salt);
    return salt;
}
```

Generierung des Salts mittels SecureRandom

```java
private static final int ITERATION_COUNT = 65536;
```

Einstellung der Kosten

```java
private static final int KEY_LENGTH = 512;
private static final String CRYPTO = "PBKDF2WithHmacSHA512";
```

Hashfunktion und Schlüssel-länge

```java
public byte[] generateHash(String password, byte[] salt) {
    KeySpec spec = new PBEKeySpec(
        password.toCharArray(), salt, ITERATION_COUNT, KEY_LENGTH);
    SecretKeyFactory factory = SecretKeyFactory.getInstance(CYRPTO);
    return factory.generateSecret(spec).getEncoded();
}
```

Generierung des Hashs

Beispiel: Erstellung eines Passwortrecords / Passwordprüfung

```
public static class PasswordRecord {
    private final byte[] hash;
    private final byte[] salt;

    public PasswordRecord(byte[] hash, byte[] salt) {
        this.hash = hash;
        this.salt = salt;
    }
```

Record enthält mindestens Hash und Salt. Ideal wäre noch Parameter der Hashfunktion zu speichern.

```
public PasswordRecord create(String password) {
    byte[] salt = generateSalt();
    byte[] hash = generateHash(password, salt);
    return new PasswordRecord(hash, salt);
}
```

1. Salt erstellen.
2. Hash mit Password und Salt erstellen.
3. Record mit Hash und Salt zurückgeben.

```
public boolean compare(String password, PasswordRecord record) {
    byte[] newPasswordHash = generateHash(password, record.salt);
    return Arrays.equals(newPasswordHash, record.hash);
}
```

Erstellung eines Passwordhash mit eingegebenem Passwort und Salt aus dem Record. Anschliessend Vergleich.

Autorisierung

– Nach Verknüpfung mit Konto ist eine Session autorisiert auf bestimmte Ressourcen oder Funktionalitäten zuzugreifen:

```
if (session.hasRole("logger")) {
    while (true) {
        LogMessage msg = receiveLogMsg();
        ...
    }
}
```

Typische Verfahren zur Zugriffkontrolle:

– **Role Based Access Control**: Definition von Rollen (bspw. Verkäufer, Buchhalter, Manager, Mechaniker, …) und prüfen, ob Benutzersession für die benötigte Rolle die Rechte hat.

– **Row Level Security**: Funktionalität von Datenbanken: Definiert pro Ressource (oft in Table-Row gespeichert), wer darauf zugreifen darf.

Zusammenfassung

- Sichere Kommunikation in verteilten Systemen: Zugriffschutz, Manipulationssicherheit, Abhörsicherheit Nachvollziehbarkeit.
- TLS-Protokoll: Transportverschlüsselung implementiert als Anwendungsprotokoll.
- Zertifikate authentifizieren das Zielsystem und werden von vertrauenswürdigen Zertifizierungsstellen erstellt.
- Eine Benutzersession ist eine zeitlich beschränkte Zweiwege-Kommunikation.
- Authentifizierung eines Client findet in der Regel auf Basis einer Benutzersession statt.
- Bei einer Authentifizierung mittels Passwort muss dieses sicher gespeichert werden.
- Mittels Autorisierung wird sichergestellt, dass nur berechtige Benutzer auf Ressourcen und Funktionalität zugreifen.

Literatur

- Distributed Systems (3rd Edition), Maarten van Steen, Andrew S. Tanenbaum, Verleger: Maarten van Steen (ehemals Pearson Education Inc.), 2017.

Fragen?