Sichere Kommunikation

Rückblick

- ✓ Sicheres Passwort
- ✓ Zwei-Faktor-Authentisierung

- ✓ Bruteforce Schutz
- ✓ Passwort-Hashing
- ✓ Salt/Pepper

Kommunikation im Internet

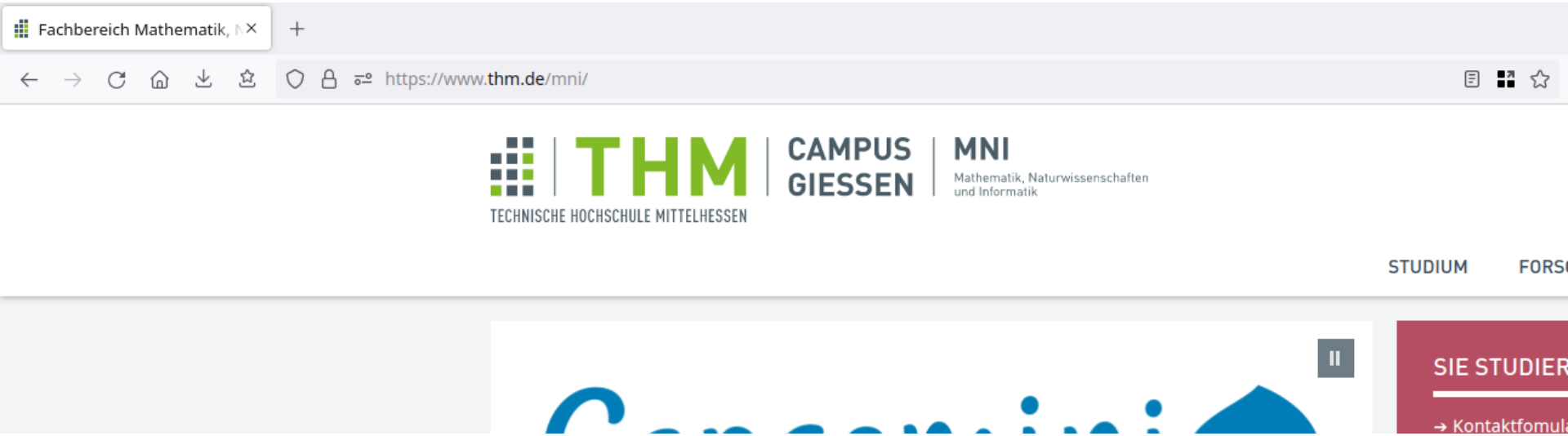
- HTTP ist ein Protokoll der Anwendungsschicht und nutzt TCP als Transportprotokoll.
- Internet-Anwendungen kommunizieren über HTTP.

Adressierung von Webinhalten

- Zugriff auf WWW-Seiten erfordert Klärung:
  - Wie lautet der lokale Name der Seite beim Server?
  - Wo liegt die Seite?
  - Wie kann auf die Seite zugegriffen werden?

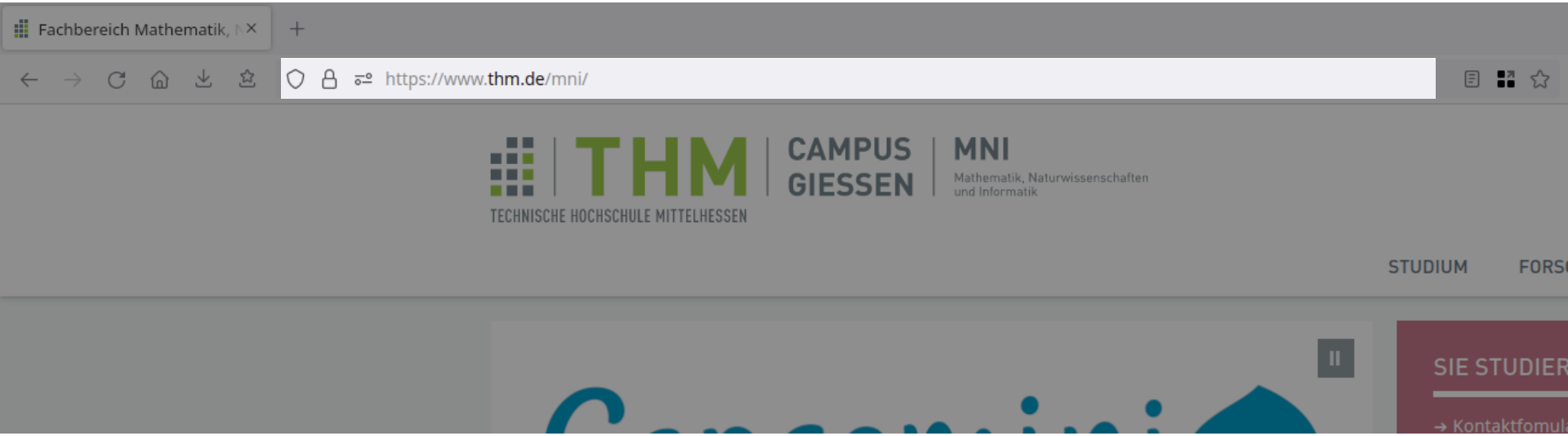
> Uniform Resource Locator

Uniform Resource Locator



Uniform Resource Locator

- Aufgabe: Angabe der Position einer Ressource



Uniform Resource Locator

- Protokoll
- Z.B.: http, https, ftp

http://name:pass@mni.thm.de:443

Uniform Resource Locator

- Authentifikation

http://name:pass@mni.thm.de:443

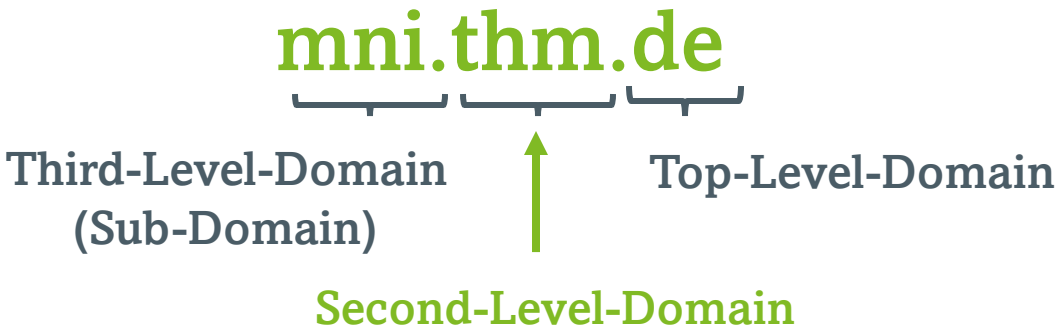
Uniform Resource Locator

- Domain

http://name:pass@mni.thm.de:443

Uniform Resource Locator

- Domain



Uniform Resource Locator

- Domain Name System (DNS)

Uniform Resource Locator

- Port
- Standardports: HTTP: 80/TCP, HTTPS: 443/TCP

http://name:pass@mni.thm.de:443

Uniform Resource Locator

- Aufgabe: Angabe der Position einer Ressource (meist Datei)

http://name:pass@mni.thm.de:443/moodle/course/view.php?id=8#th2

Uniform Resource Locator

- Pfad auf Server

/moodle/course/view.php?id=8#th2

Uniform Resource Locator

- Parameter

/moodle/course/view.php?id=8#th2

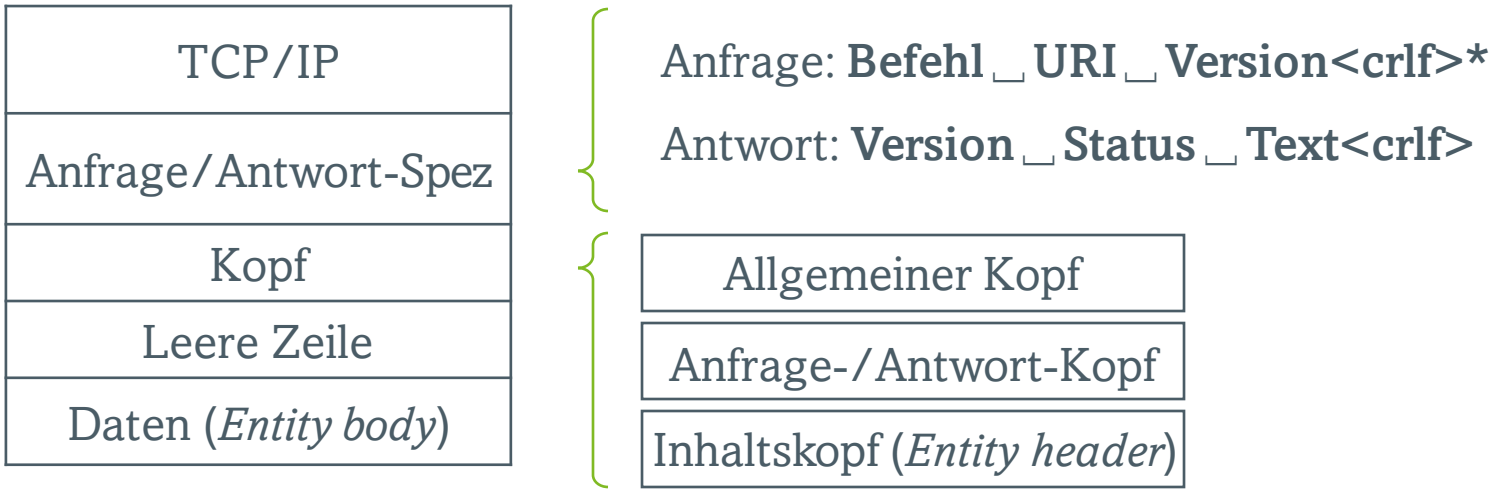
Uniform Resource Locator

- Fragment

/moodle/course/view.php?id=8#th2

Http

- Unterscheidung zweier Nachrichtentypen
  - Request (Anfrage)
  - Response (Antwort)
- Aufbau einer HTTP-Nachricht



*<crlf> sind die beiden Zeichen mit ASCII-Code 13 und 10

HTTP Anfrage

- **Befehle (auch Methoden genannt)**
  - GET Lesen einer Seite (allg.: einer Ressource)
  - HEAD Lesen eines Seitenkopfs (insbesondere Angaben, wann die Seite zuletzt geändert wurde)
  - PUT Abspeichern einer Seite (Neuanlegen)
  - POST Anhangen von Daten (wird zum Übertragen von Formulardaten verwendet)
  - DELETE Löschen einer Seite (sofern erlaubt)

HTTP Anfrage

Beispiel:

```
GET /site/ HTTP/2
Host: www.thm.de
User-Agent: Mozilla/5.0 (Windows NT 10.0;…) … Firefox/112.0
Accept: text/html,application/xhtml+xml,…,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content.Length: 345
```

Request headers

General headers

Representation headers

HTTP Antwort

- **Antwort besteht aus**
  - Statuszeile (Dreistelliger Zahlencode mit Klartexthinweis)
  - Kopf
  - Daten

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 19 Apr 2023 14:23:15 GMT
Etag: „d9b3b803e9a0def76ef89f0ac7e7“
Keep-Alive: timeout=5, max=999
Last-Modified: Wed, 19 Apr 2023 10:12:48 GMT
Server: Apache
Set-Cookie: csrftoken=…
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding
X-Frame-Options: DENY
```

Request headers

General headers

Representation headers

HTTP Antwort

200 OK
201 Created
202 Accepted
204 No content

2xx Success

301 Moved permanently
307 Temporary redirect

3xx Redirection

400 Bad request
401 Unauthorized
402 Payment required
403 Forbidden
404 Not found

4xx Client error

500 Internal server error
503 Service unavailable

5xx Server error

HTTP und Sitzungen

HTTP ist nicht sitzungsorientiert

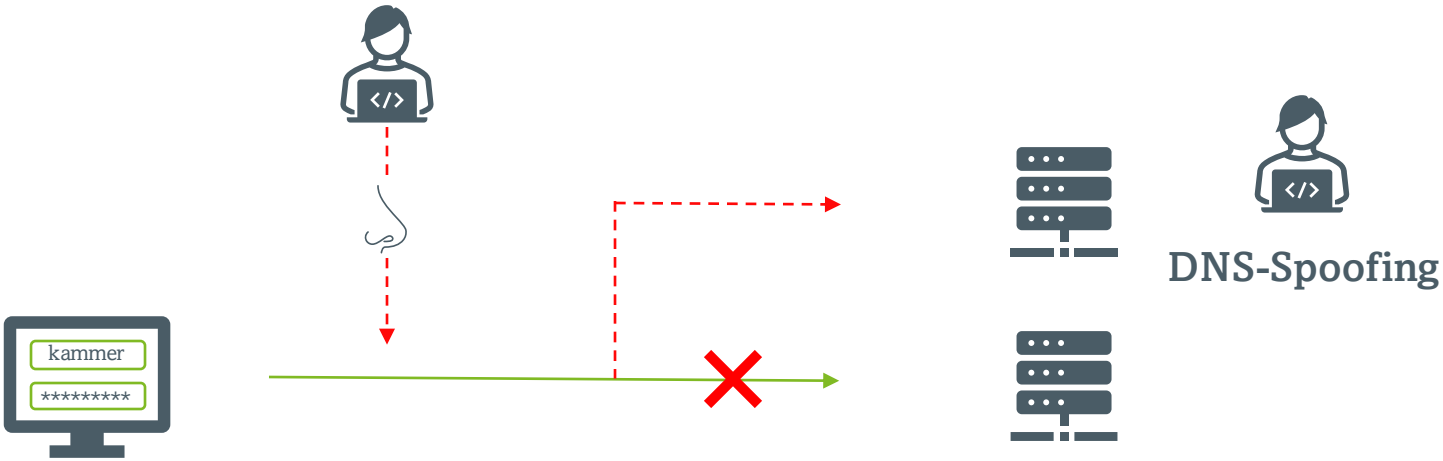
- Jede HTTP-Abfrage ist eine eigene Verbindungsaufnahme
- Bem.: Es ist möglich im Kopf anzugeben, eine Verbindung über ein paar Anfragen hinweg wiederzuverwenden (Keep-alive)
- Jedoch ist jede Anfrage völlig eigenständig und hat nichts mit einer vorherigen Anfrage zu tun

Problem bei geschützten Seiten

- Wie merkt sich das System den Login?

- Erhaltung einer Sitzung auf Applikationsebene beim Klienten und Server
- Sitzungsreferenzen über URL bzw. Cookies

Sicherheitslücken http



- ✓ Sicheres Passwort
- ✓ Zwei-Faktor-Authentisierung

- ✓ Bruteforce Schutz
- ✓ Passwort-Hashing
- ✓ Salt/Pepper

ℹ Man-in-the-middle Attacke

Netzwerkanalyse

- **Traffic mitzuloggen und ggf. auswerten**
- **Programme: tcpdump, Wireshark**

Beispiel:

```
> ifconfig
eno167777 Link encap:Ethernet Hardware Adresse 00:0C:29:1C:FE:67
      inet Adresse:192.168.121.128 Bcast:192.168.121.255/24
      inet6 Adresse: fe80::20c:29ff:fe1c:fe67/64
Gültigkeitsbereich: Verbindung
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:3555 errors:0 dropped:0 overruns:0 frame:0
      TX packets:2938 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 Sendewarteschlangenlänge:1000
      RX bytes:1466622 (1.3 Mb) TX bytes:494421 (482.8 Kb)
> tcpdump –i eno167777
```

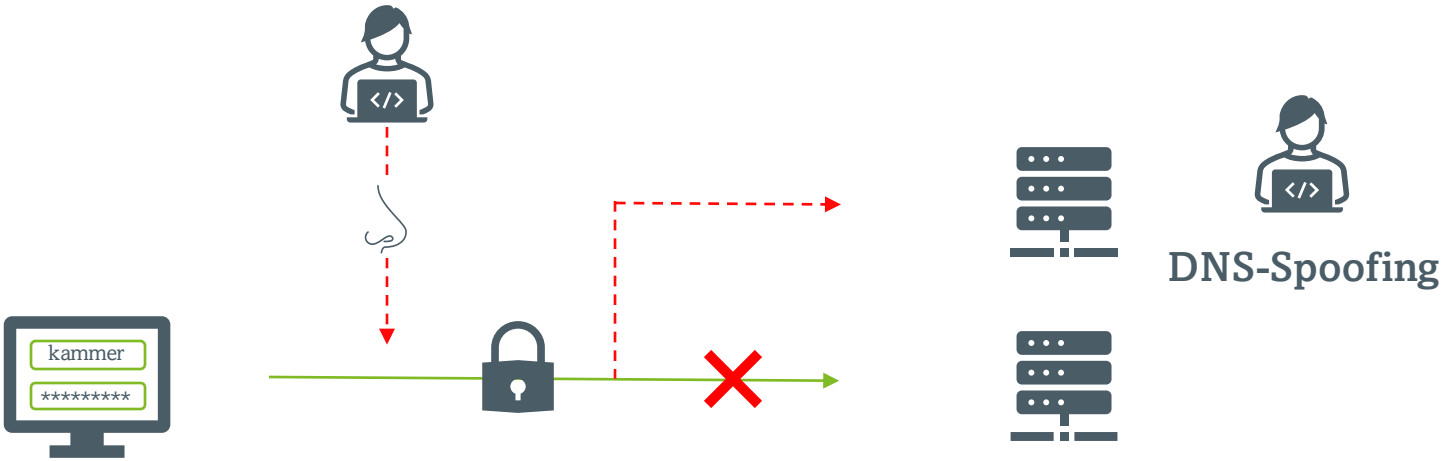
DNS-Spoofing

- Auflösen der URL zu einer falschen IP-Adresse

~~google.com~~	~~142.250.65.206~~
google.com	101.210.30.145

- Varianten:
  - Angriff auf den Client oder lokalen Router
  - Angriff auf die Antwort des DNS-Servers
  - Angriff auf den DNS-Server

Sicherheitslücken http



- ✓ Sicheres Passwort
- ✓ Zwei-Faktor-Authentisierung

- ✓ Bruteforce Schutz
- ✓ Passwort-Hashing
- ✓ Salt/Pepper

Wie erreicht man sichere Kommunikation?

→ 2 Methoden

- **Verschlüsselung von Daten (Encryption)**
  - Klartext (Plain) → Geheimtext (Cipher)
  - Inhalt ist nur dem Empfänger zugänglich

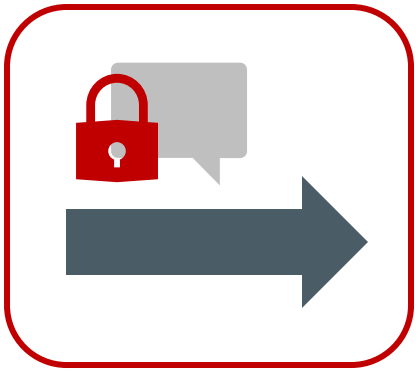
- **Signieren von Daten (Signing)**
  - Prüfsumme (der digitalen Signatur) von Plain / Cipher
  - Ermöglicht eine Verifizierung

Verschlüsselung von Daten (Encryption)

Klartext (Plain)

Geheimtext (Cipher)

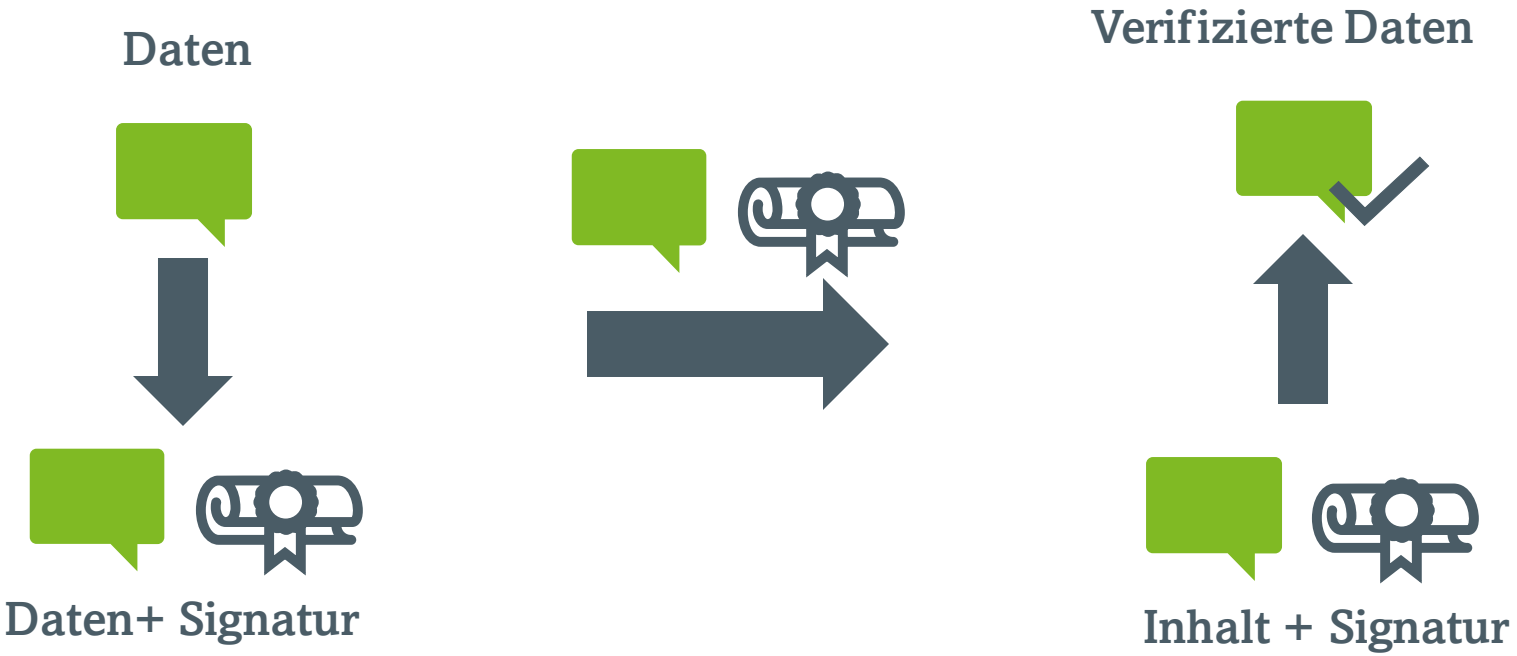
Nur für Empfänger zugänglich



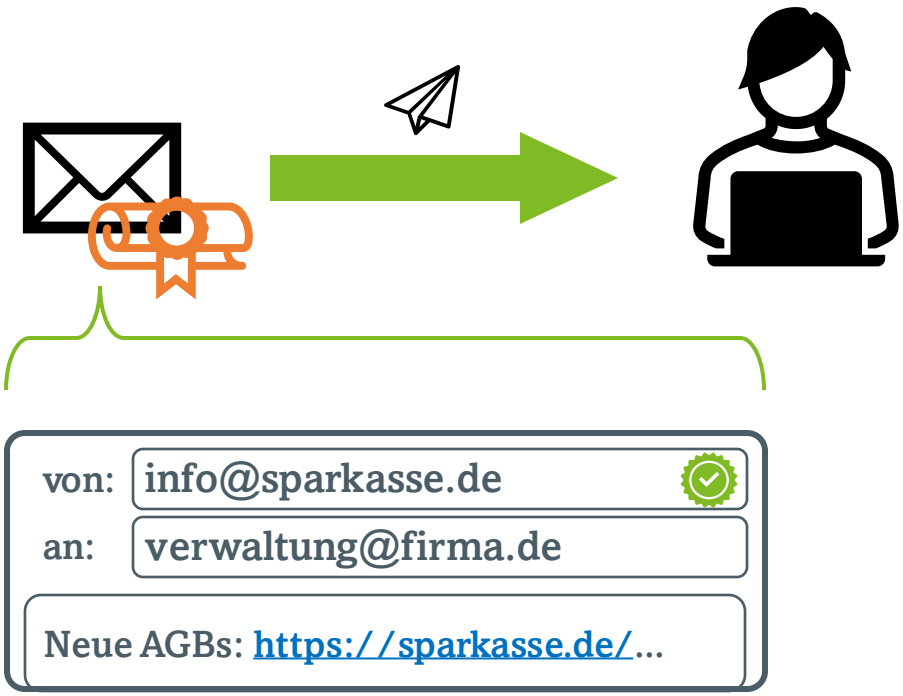
Klartext (Plain)

Geheimtext (Cipher)

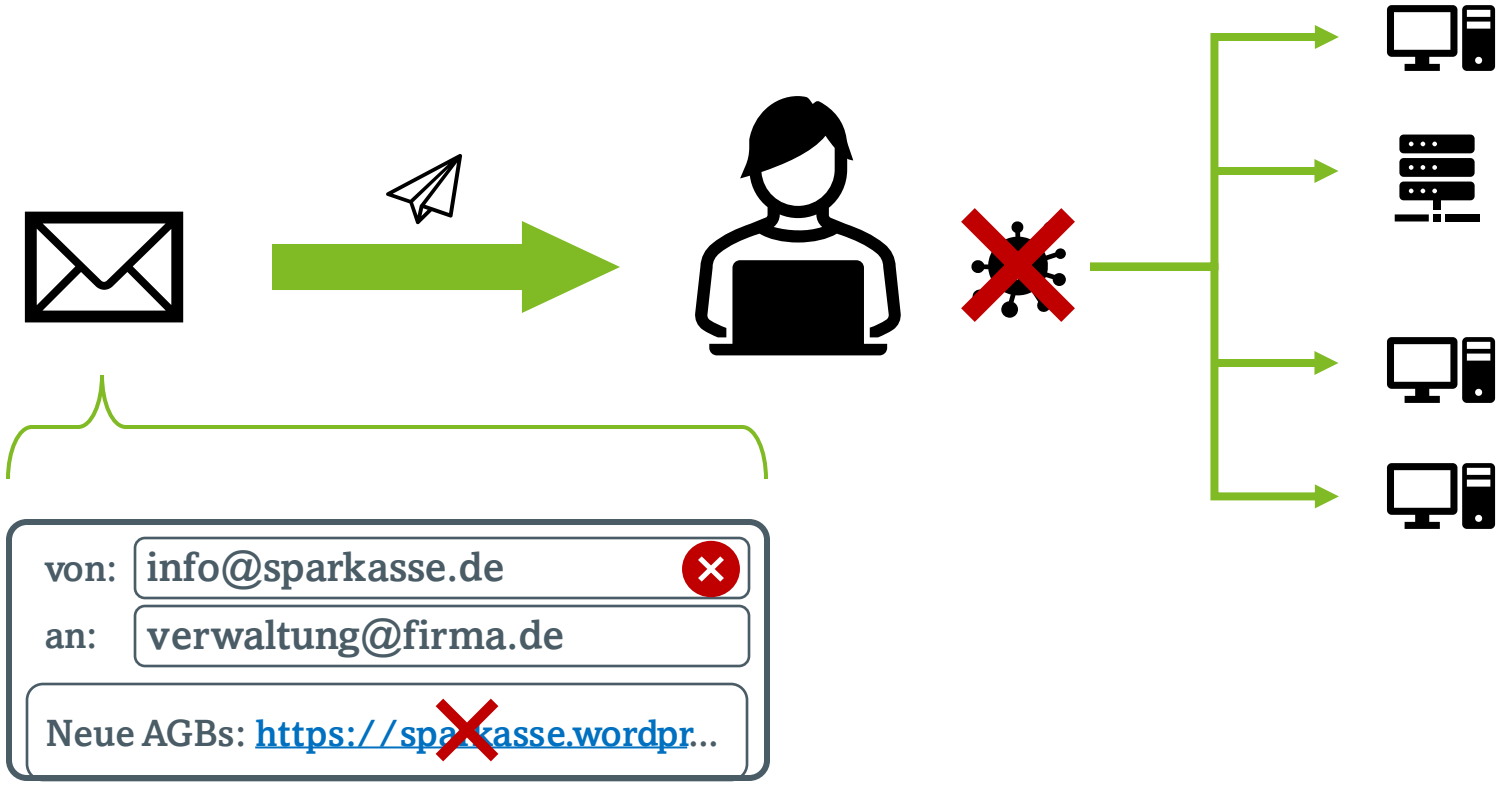
Signieren von Daten (Signing)



Rückblick: Mail-Spoofing



Rückblick: Mail-Spoofing



Wie erreicht man sichere Kommunikation?

→ 2 Methoden

- **Verschlüsselung von Daten (Encryption)**
  - Klartext (Plain) → Geheimtext (Cipher)
  - Inhalt ist nur dem Empfänger zugänglich

- **Signieren von Daten (Signing)**
  - Prüfsumme (der digitalen Signatur) von Plain / Cipher
  - Ermöglicht eine Verifizierung

→ Einige kryptographische Algorithmen können beides

Klassifikation kryptographischer Algorithmen

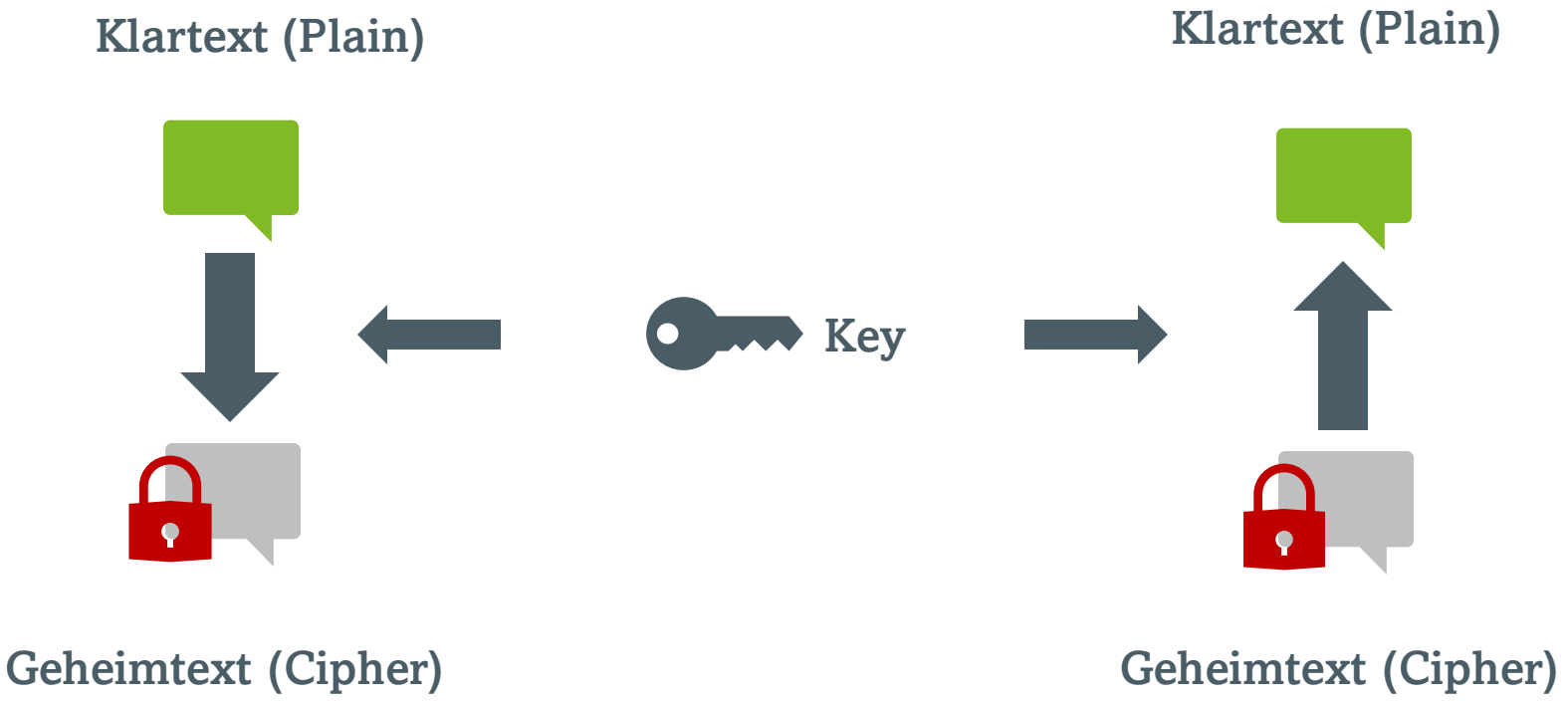
Klassifikation kryptographischer Algorithmen

- Kryptographische Hash-Funktionen
  - Benutzen keinen Schlüssel

Klassifikation kryptographischer Algorithmen

- Kryptographische Hash-Funktionen
  - Benutzen keinen Schlüssel
- Symmetrische Verfahren
  - Benutzen einen Schlüssel für Ver- und Entschlüsselung, Signierung und Verifikation

Symmetrische Verfahren



Symmetrische Verfahren

- Gewünschte Eigenschaften:
  - Einfach zu spezifizieren, zu verstehen und zu implementieren
  - Anwendbar für verschiedenste Probleme
  - Lauffähig auf vielen Geräten
  - Effizient in der Nutzung
  - Funktionalität beweisbar
  - Hohes Level an Sicherheit
  - **Sicherheit kommt nur durch den Schlüssel**

> *„Es darf nicht der Geheimhaltung bedürfen und soll ohne Schaden in Feindeshand fallen können."*
> [Auguste Kerckhoffs, La cryptographie militaire 1883]

Symmetrische Algorithmen

- DES (Data Encryption Standard)
- 3DES
- RC4
- AES (Advanced Encryption Standard)

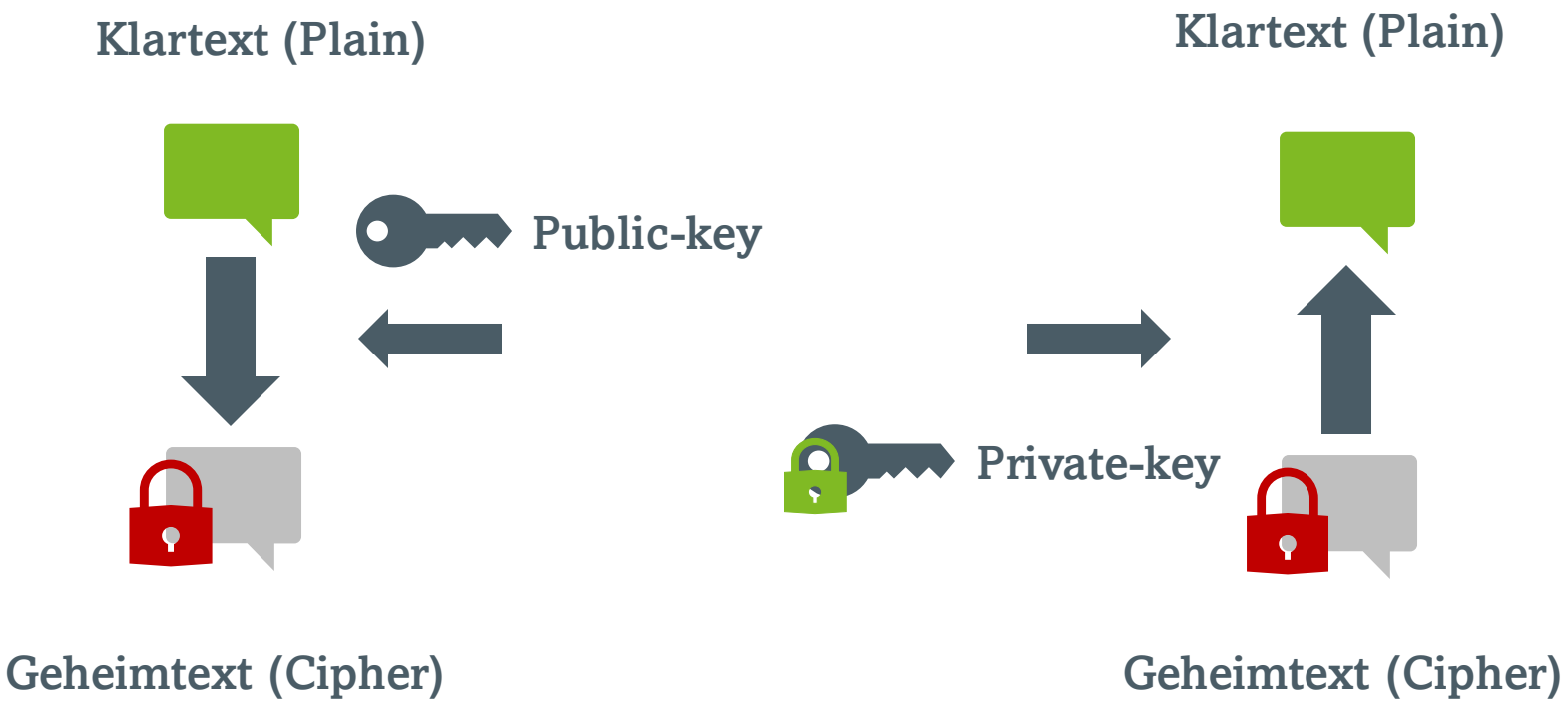
Klassifikation kryptographischer Algorithmen

- Kryptographische Hash-Funktionen
  - Benutzen keinen Schlüssel
- Symmetrische Verfahren
  - Benutzen einen Schlüssel für Ver- und Entschlüsselung, Signierung und Verifikation
- Asymmetrische Verfahren
  - Benutzen zwei Schlüssel (public und private) für Ver- und Entschlüsselung, Signierung und Verifikation

Asymmetrische Algorithmen - Idee

- **Verwenden von zwei Schlüsseln (K_{priv} und K_{pub}) zur Ver- (E) und Entschlüsselung (D)**
- **Der Schlüssel K_{priv} ist nur A bekannt**
- **K_{pub} ist der öffentliche Schlüssel von A und allen bekannt**
- **Gegeben $c = E(K_{pub}, m)$ und K_{pub}**
  - $\rightarrow$ Es soll unmöglich $m = D(K_{priv}, c)$ zu berechnen.
  - $\rightarrow$ Insbesondere soll man von K_{pub} nicht auf K_{priv} schließen können

Asymmetrische Algorithmen - Idee



Asymmetrische Algorithmen - Anwendungen

- **Verschlüsselung:**
  - B kann eine Nachricht mit K_{pub} von A verschlüsseln und nur A kann es mit K_{priv} entschlüsseln
- **Signierung:**
  - Nur A kann eine Nachricht mit K_{priv} verschlüsseln. Jeder kann mit K_{pub} aber überprüfen, dass die Nachricht von A ist.

Jeder muss überprüfen können, dass K_{pub} wirklich der öffentliche Schlüssel von A ist!

Asymmetrische Algorithmen

Was eignet sich für einen asymmetrischen Algorithmus?

- Probleme der Mathematik/Informatik, die schwer zu lösen sind, wenn man nur K_{pub} , aber einfach, wenn man auch K_{priv} kennt

Diskretes Logarithmus Problem: Diffie-Hellman, ElGamal, DAS

Faktorisierung: RSA

Probleme von Asymmetrischen Verfahren

Asymmetrische Verfahren sind im Vergleich zu symmetrischen sehr langsam.

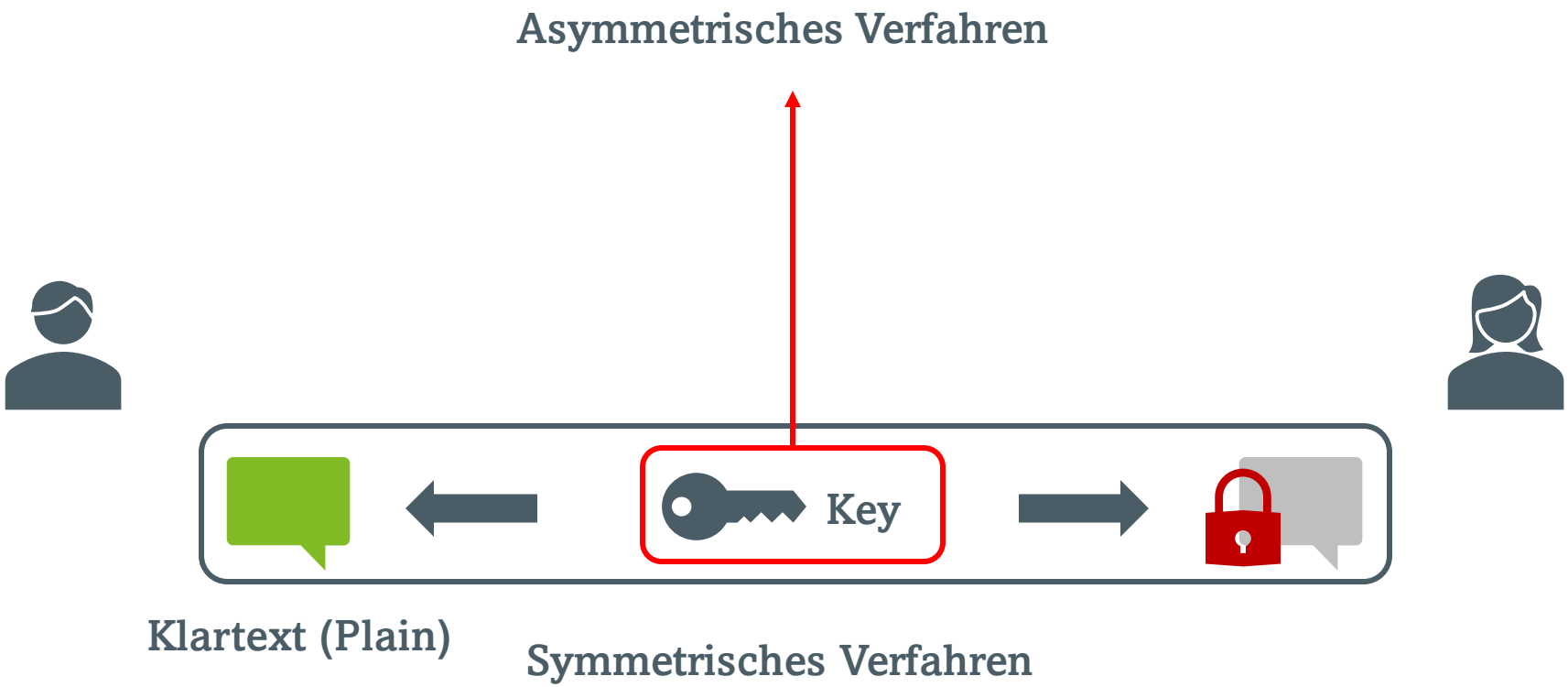
Dennoch möchte man gerne die asymmetrischen Verfahren nutzen.

Symmetrische Verfahren erzwingen ein Teilen der Schlüssel

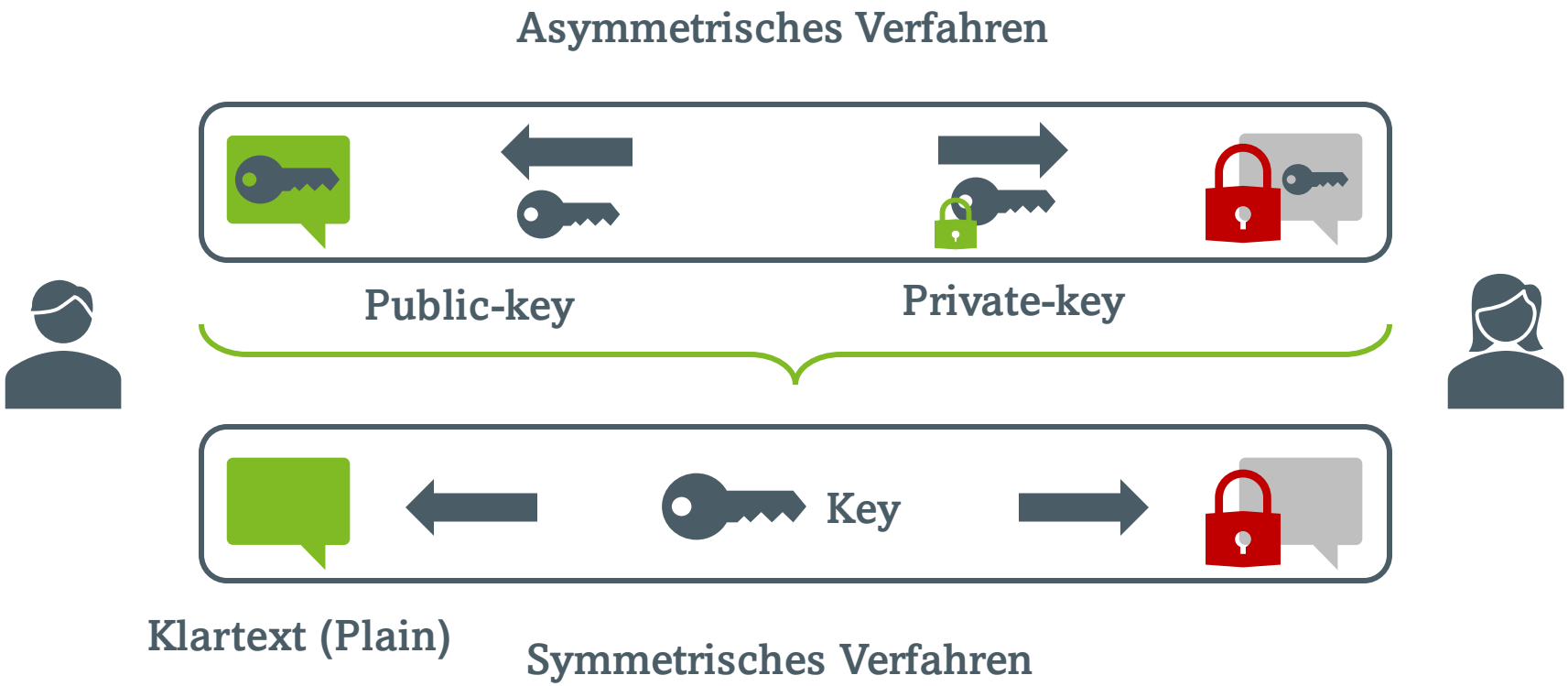
Asymmetrische Verfahren ermöglichen eine digitale Signatur

Kombination beider Methoden

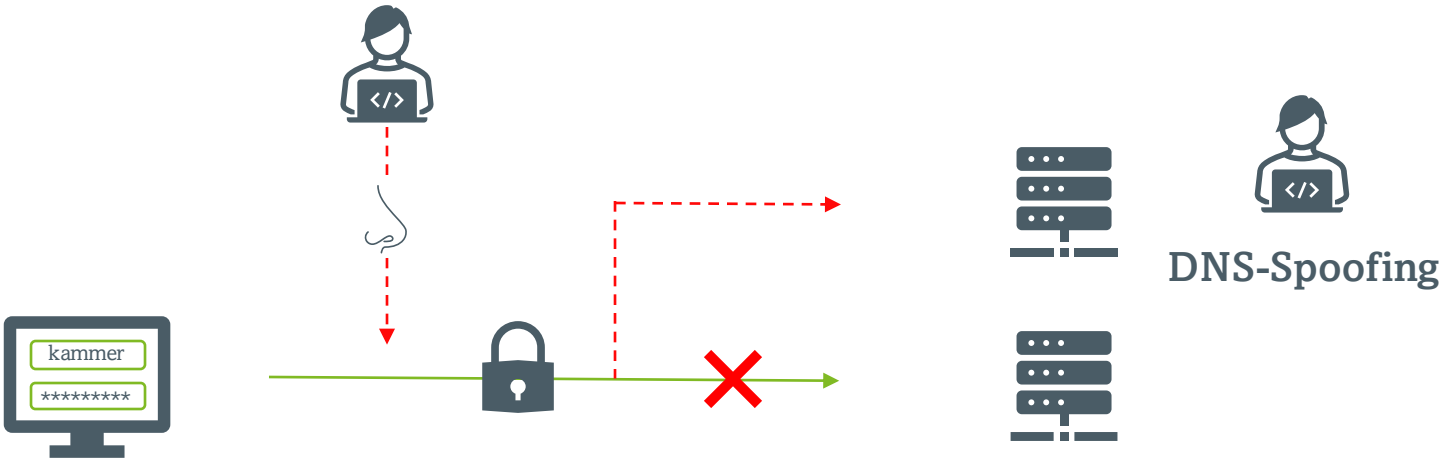
Kombination der Algorithmen



Kombination der Algorithmen



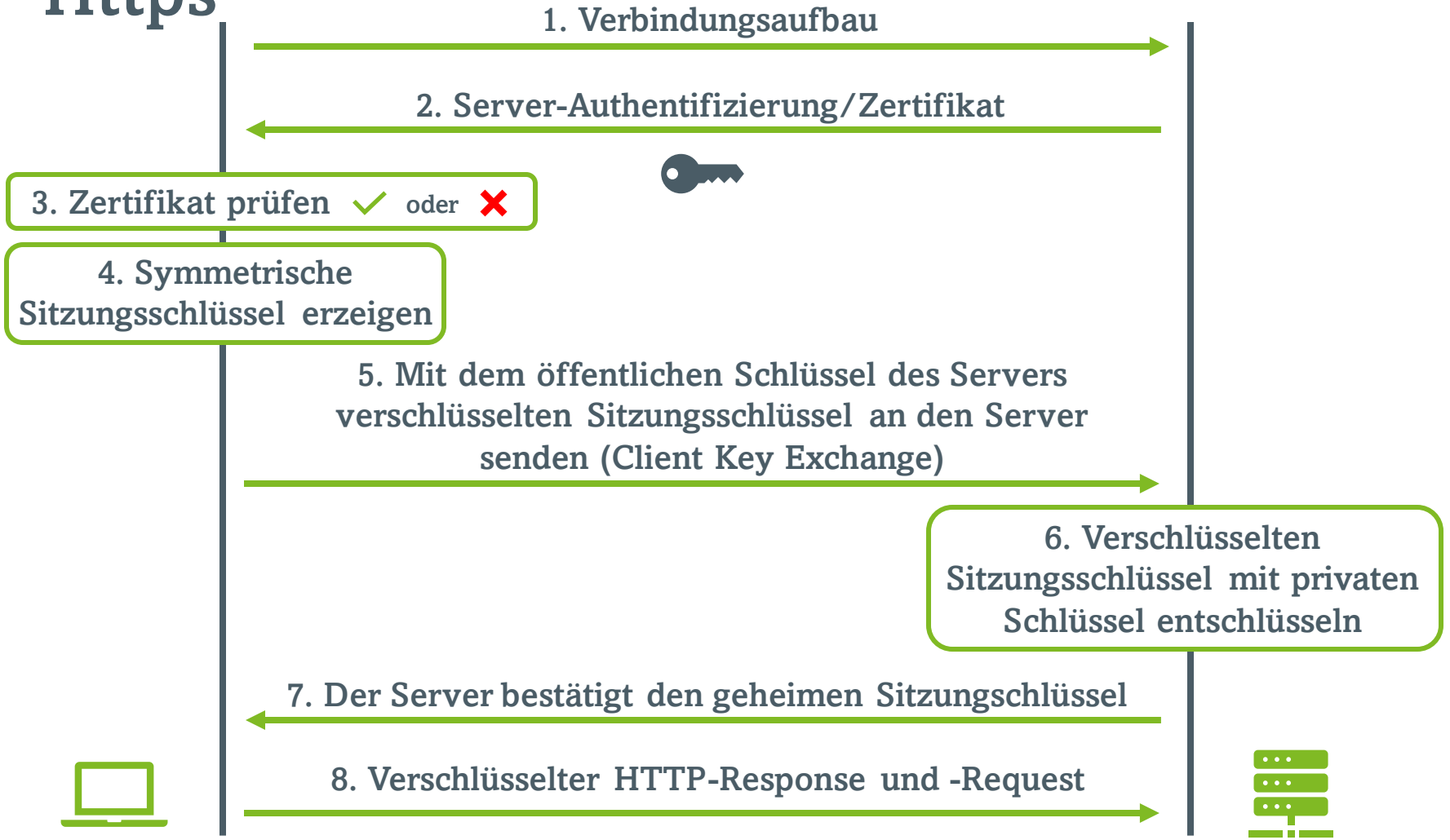
Sicherheitslücken http



- ✓ Bruteforce Schutz
- ✓ Sicheres Passwort
- ✓ Zwei-Faktor-Authentisierung

- ✓ Passwort-Hashing
- ✓ Salt/Pepper

Https


1. Verbindungsaufbau
2. Server-Authentifizierung/Zertifikat
3. Zertifikat prüfen ✓ oder ✗
4. Symmetrische Sitzungsschlüssel erzeigen
5. Mit dem öffentlichen Schlüssel des Servers verschlüsselten Sitzungsschlüssel an den Server senden (Client Key Exchange)
6. Verschlüsselten Sitzungsschlüssel mit privaten Schlüssel entschlüsseln
7. Der Server bestätigt den geheimen Sitzungschlüssel
8. Verschlüsselter HTTP-Response und -Request

Certificate Authority (CA)

- **Es ist wichtig sicherzustellen, dass der richtige öffentliche Schlüssel verwendet wird.**

- **Zertifizierungsstelle für digitale Zertifikate**

Die Stelle beglaubigt die Identität, indem sie den öffentlichen Schlüssel mit Ihrer eigenen digitalen Unterschrift versieht und so ein sog. Digitalen Zertifikat erstellt. Die Stelle bildet den Kern der Public-Key-Infrastruktur und trägt dabei

- **die Verantwortung für die Bereitstellung**
- **Zuweisung und Integritätssicherung der von ihr ausgegebenen Zertifikate**

Certificate Authority (CA)

Die digitalen Zertifikate enthalten

- Schlüssel und Gültigkeitsdauer
- Zertifikatskette
- Verweise auf Zertifikatsperrlisten

Aufgaben einer CA

Generierung von Zertifikaten (Certificate Issuance)

- Erzeugung der Datenstrukturen und Signatur

Speicherung (Certification Repository)

- Allgemein zugängliches Repository für Zertifikate

Widerruf und Sperrung (Certificate Revocation)

- Z.B. falls geheimer Schlüssel kompromittiert wurde

Aktualisierung (Certifcation Update)

- Erneuerung des Zertifikates nach Ablauf der Gültigkeit

Schlüsselerzeugung (Key Generation)

Aufgaben einer CA

Historienverwaltung (Certification History)

- Speicherung nicht mehr gültiger Zertifikate (zur Beweissicherung)

Beglaubigung (Notarization)

- CA signiert Vorgänge zwischen Benutzern (z.B. Verträge)

Zeitstempeldienst (Time Stamping)

- CA bindet Info an Zeit (Gültigkeit)

Realm-übergreifende Zertifizierung (Cross-Certification)

- Eigene CA zertifiziert fremde CAs

Attribut-Zertifikate (Attribute Certificate)

- Binden von Attributen an eine Identität (z.B. Berechtigungen, Vollmachten, ...)

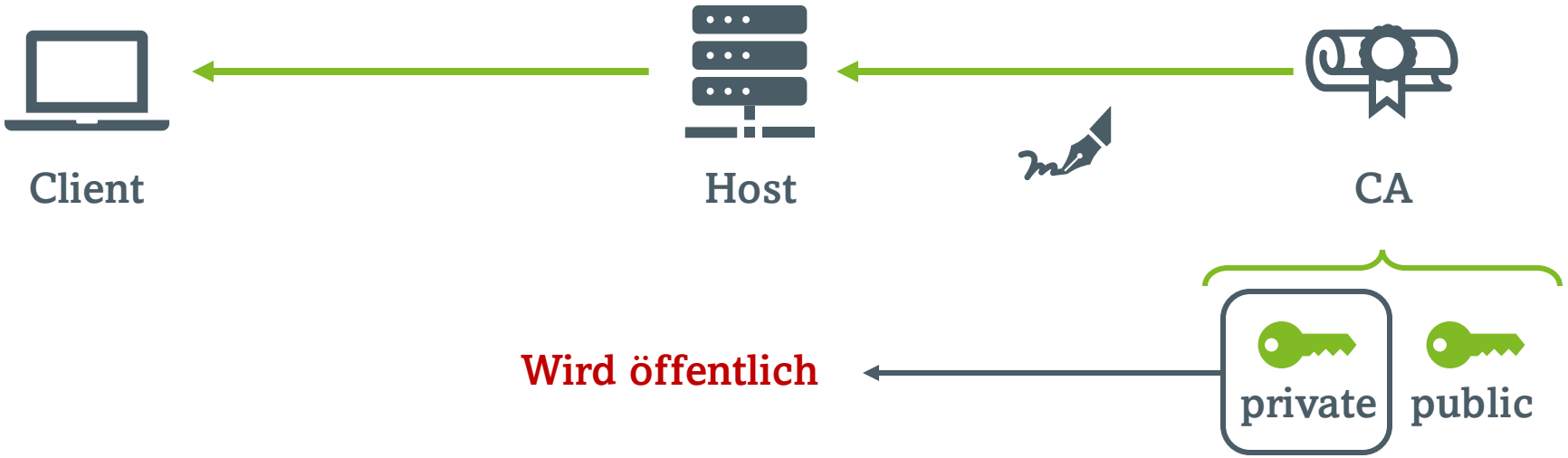
Https-Zertifikat prüfen

- Root-store beinhaltet alle public-keys von vertrauenswürdigen CAs

Chain of trust (Zertifikatskette)

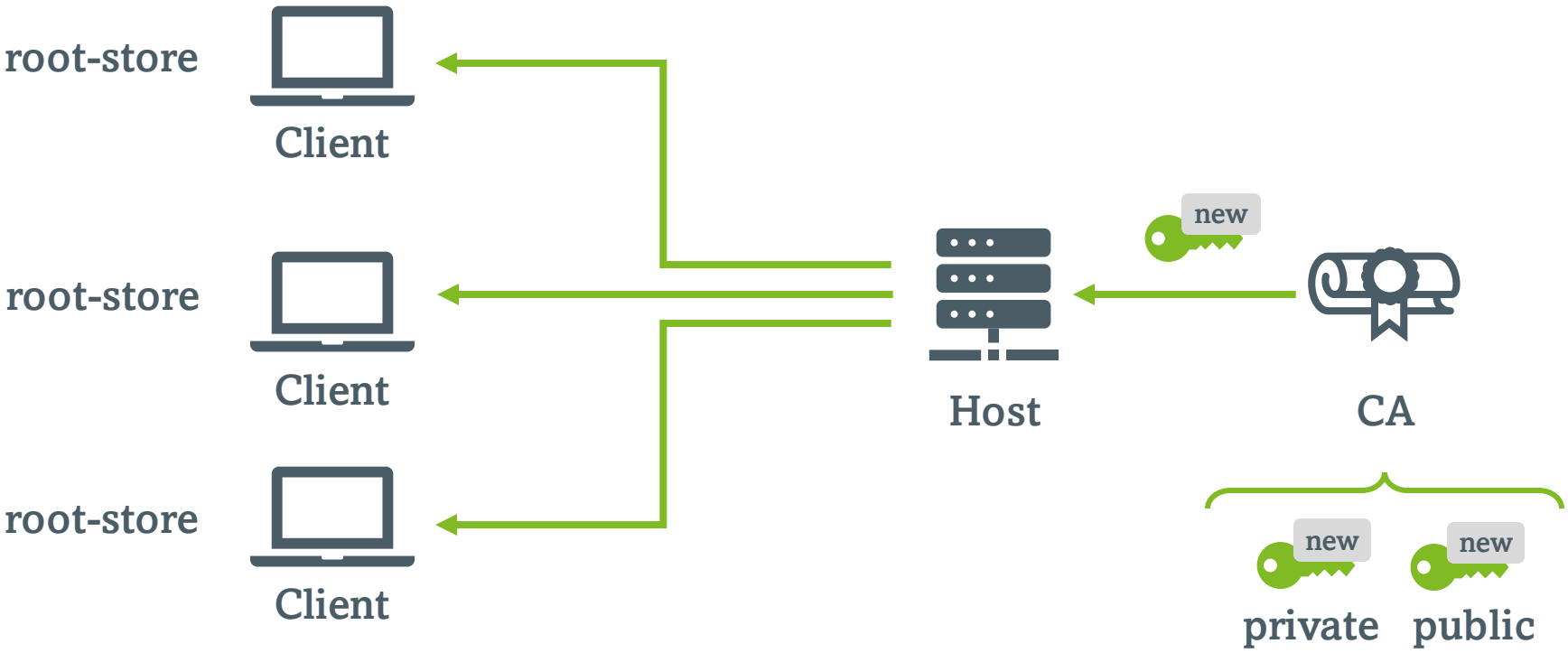
Warum benötigt man eine Zertifikatskette?

- CA unterschreibt das Zertifikat für den Host



Chain of trust (Zertifikatskette)

CA anpassen ist sehr aufwendig!

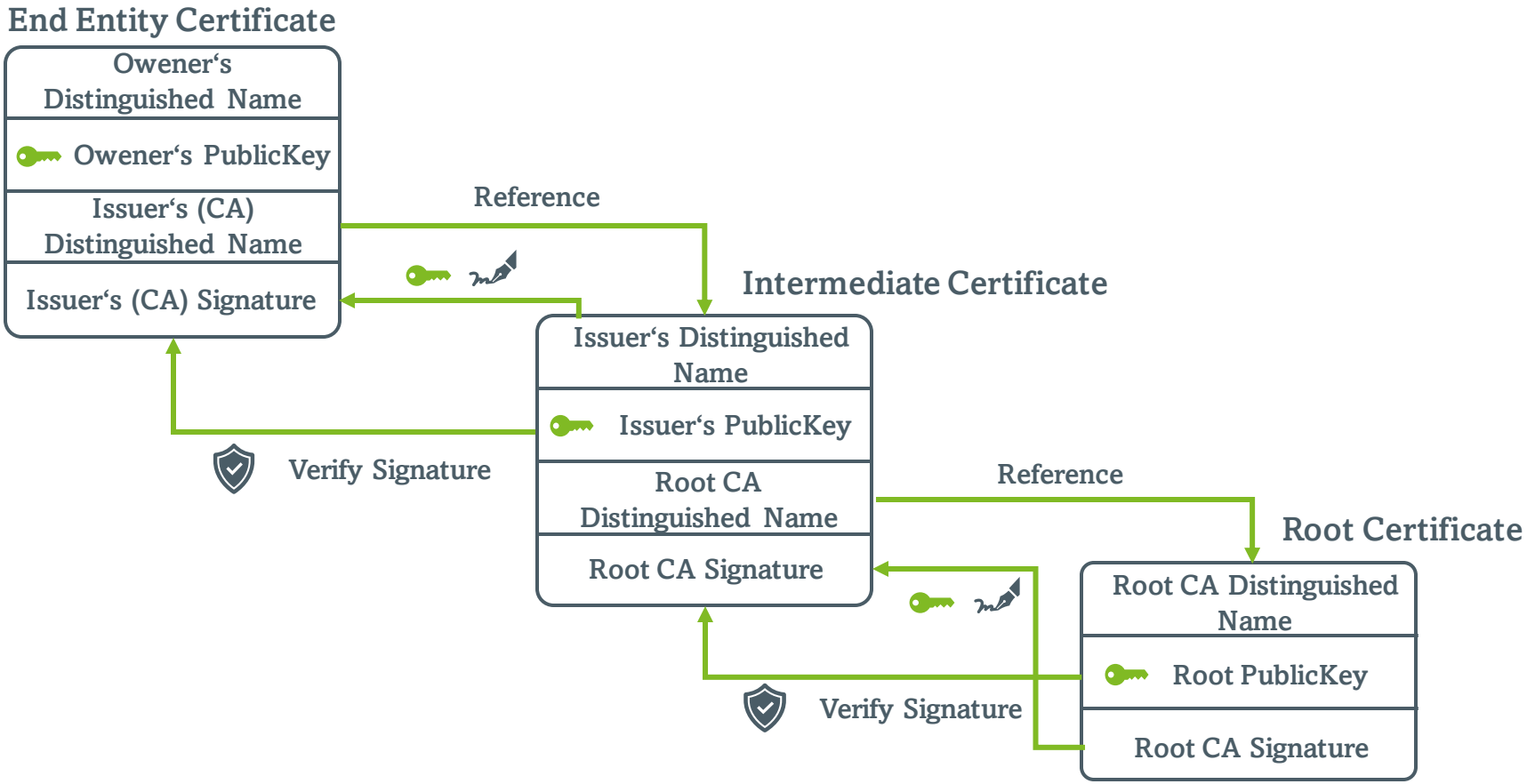


Chain of trust (Zertifikatskette)

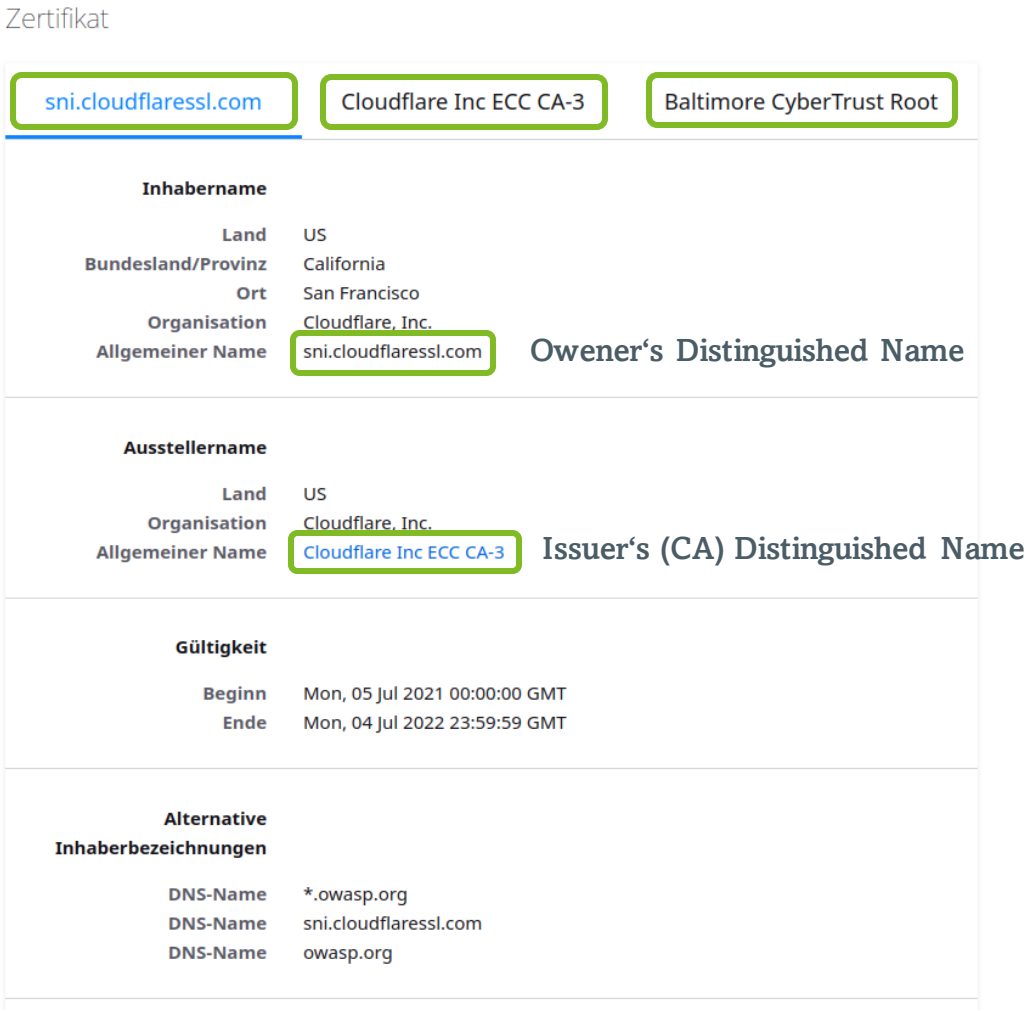
Untergeordnete CAs

Weniger müssen auf den RootCA-Key zugreifen!

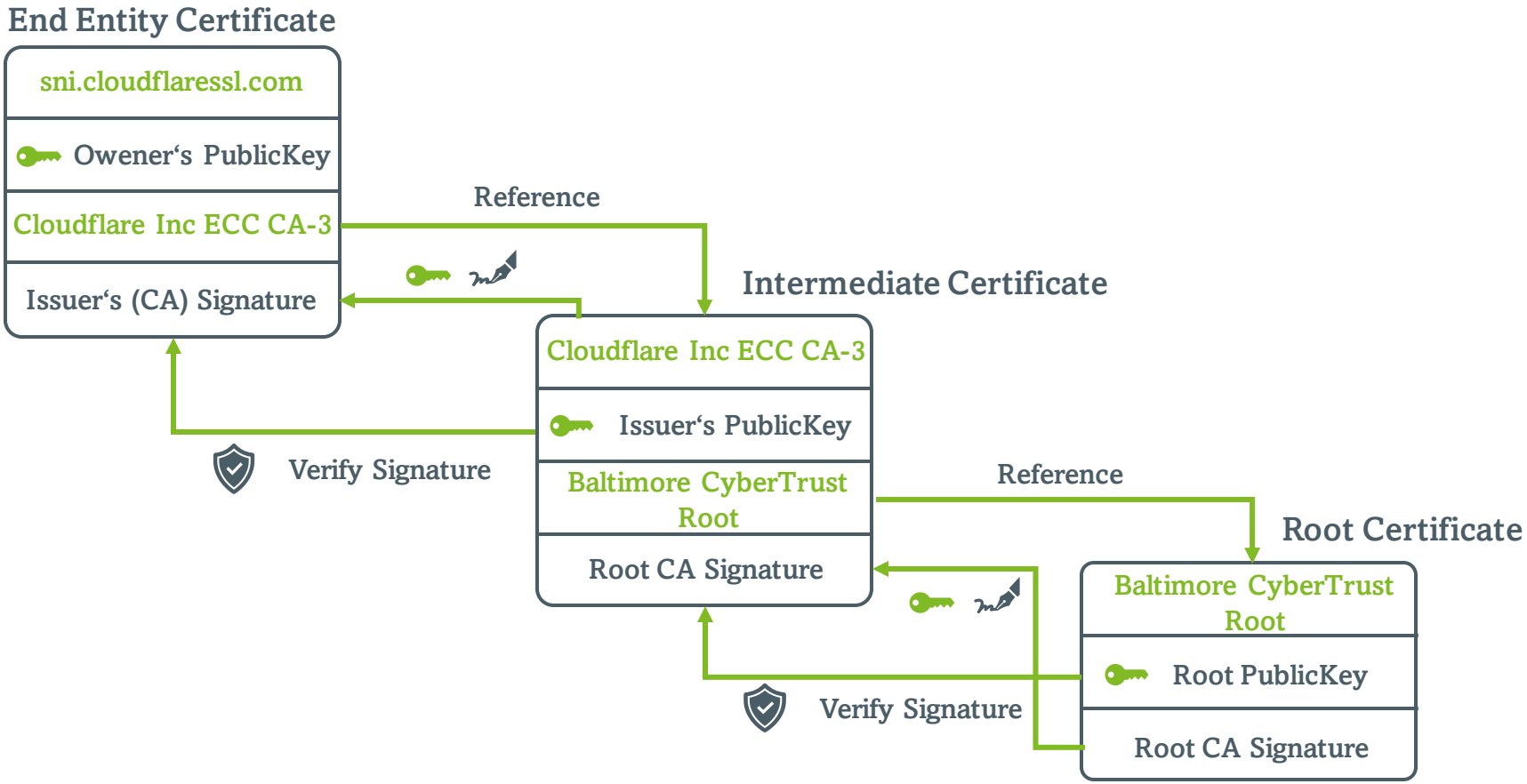
Chain of trust (Zertifikatskette)



Chain of trust (Zertifikatskette)



Chain of trust (Zertifikatskette)



Https - Zertifikat

```
> openssl s_client -connect google.de:https
```

```
CONNECTED(00000003)
depth=2 C = US, O = Google Trust Services LLC, CN = GTS Root R1
verify return:1
depth=1 C = US, O = Google Trust Services LLC, CN = GTS CA 1C3
verify return:1
depth=0 CN = *.google.de
verify return:1
---
Certificate chain
 0 s:CN = *.google.de
   i:C = US, O = Google Trust Services LLC, CN = GTS CA 1C3
 1 s:C = US, O = Google Trust Services LLC, CN = GTS CA 1C3
   i:C = US, O = Google Trust Services LLC, CN = GTS Root R1
 2 s:C = US, O = Google Trust Services LLC, CN = GTS Root R1
   i:C = BE, O = GlobalSign nv-sa, OU = Root CA, CN = GlobalSign Root CA
```

Https - Zertifikat

```
Server certificate
-----BEGIN CERTIFICATE-----
MIIEjDCCA3SgAwIBAgIRAPClIxXDGYiiCoARMpD4k84wDQYJKoZIhvcNAQELBQAw
…
C6z2lhNz27NxoUlFjwGW/A==
-----END CERTIFICATE-----
subject=CN = *.google.de
issuer=C = US, O = Google Trust Services LLC, CN = GTS CA 1C3
---
SSL handshake has read 4298 bytes and written 391 bytes
Verification: OK
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 256 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
```

Zertifikat erzeugen

1. Erzeuge ein RSA Schlüsselpaar.

RSA Schlüsselpaar erzeugen

OpenSSL generiert einen Schlüssel mit einer Schlüsselstärke (letztes Argument) von bis zu 4096 Bits.

Privaten Schlüssel erzeugen

```
openssl genrsa -out fk-vv.key 2048
```

Bessere Schlüsselstärke → schwieriger zu knacken

Aktuellste Schlüsselstärke nicht immer unterstützt

Öffentliche Schlüssel extrahieren (meist nicht nötig)

```
openssl rsa -in fk-vv.key -pubout -out fk-vv.pub
```

OpenSSL

- Implementierung von u.a. HTTPS und verschiedener Verschlüsselungen
- Kommandozeilenprogramm openssl zum Beantragen, Erzeugen und Verwalten von Zertifikaten
- Das Programm stellt allgemeine kryptographische Funktionen zum Ver- und Entschlüsseln sowie diverse weitere Werkzeuge bereit
- Nach FIPS 140-2 zertifiziertes Open-Source-Programm
  - Sicherheitsstandard des National Institute of Standards and Technology (NIST)

Zertifikat erzeugen

1. Erzeuge ein RSA Schlüsselpaar.
2. Erzeuge eine Zertifikatsanfrage

Zertifikatsanfrage erzeugen

Eine Zertifikatsanfrage besteht aus

- der vollständigen Adresse des Servers.
- der vollständigen Organisationseinheit.
- dem öffentlichen Schlüssel.
- und generierten Signatur mit die Validität überprüft werden kann.

Der Befehl fragt in einem Konsolendialog nach allen obigen Daten:

```
openssl rsa -in fk-vv.key -pubout -out fk-vv.pub
```

Zertifikat erzeugen

1. Erzeuge ein RSA Schlüsselpaar.
2. Erzeuge eine Zertifikatsanfrage.
3. Zertifikat von einem Zertifikataussteller beantragen.

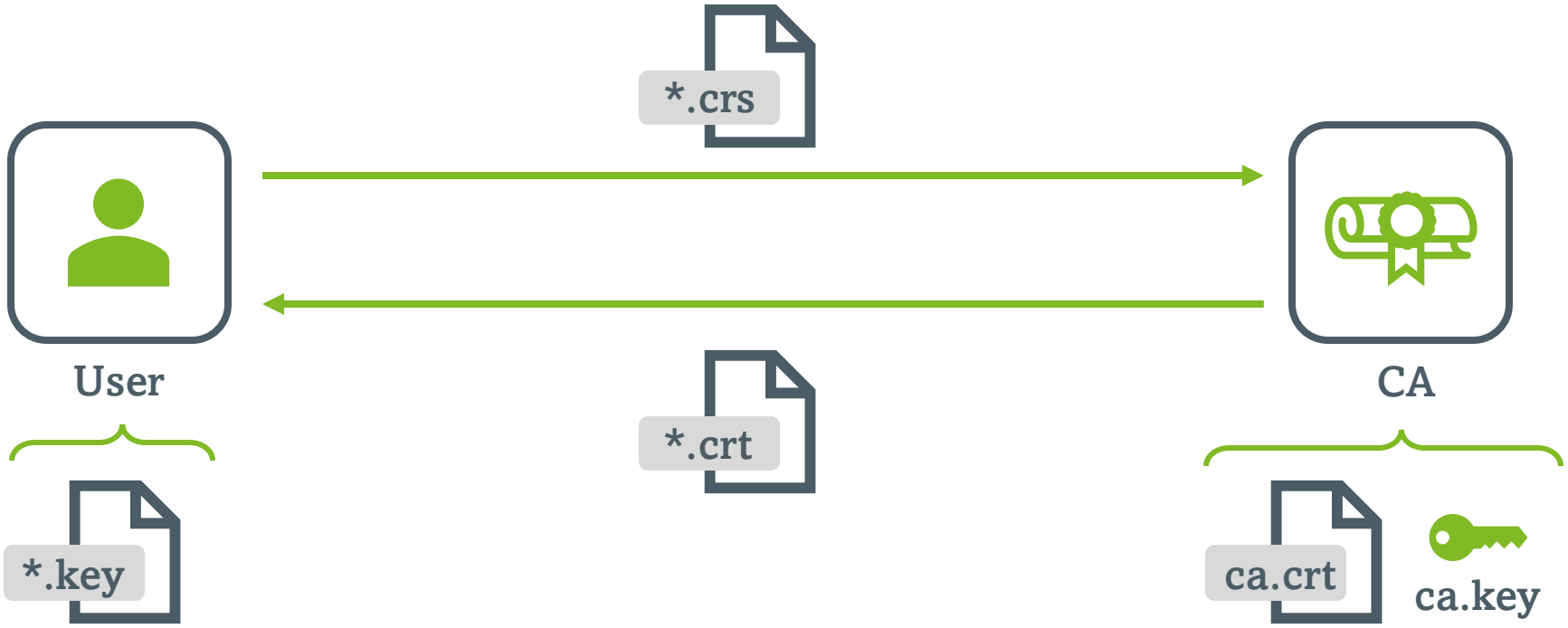
SSL Zertifikat Austeller

- **Mit einer Zertifikatsanfrage wendet man sich an einen Zertifikate-Anbieter**
  - z.B. https://ssl.de/ssl-zertifikate-anbieter.html
  - oder Let's Encrypt: https://letsencrypt.org
- **Diese erwarten die Zertifikatsanfrage (*.crs) mit einer bestimmten Schlüsselstärke.**

Der private Schlüssel sollte nie mitgeschickt werden.

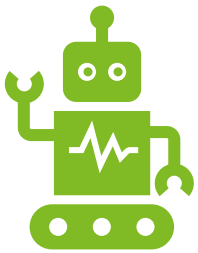
SSL Zertifikat Austeller

Der Zertifikate-Anbieter erstellt ein Zertifikat (*.crt) aus den Daten der Zertifikatsanfrage und signiert diese mit seinen Daten



Zertifikat erzeugen

1. Erzeuge ein RSA Schlüsselpaar.
2. Erzeuge eine Zertifikatsanfrage.
3. Zertifikat von einem Zertifikataussteller beantragen.



Protokoll: ACME

Verwendung:

- Certbot
- Direkte Integration in Webserver/Proxys

Selbstsignierte Zertifikate

- **Man erstellt einen CA Schlüssel**

```
openssl genrsa -out ca.key 4096
```

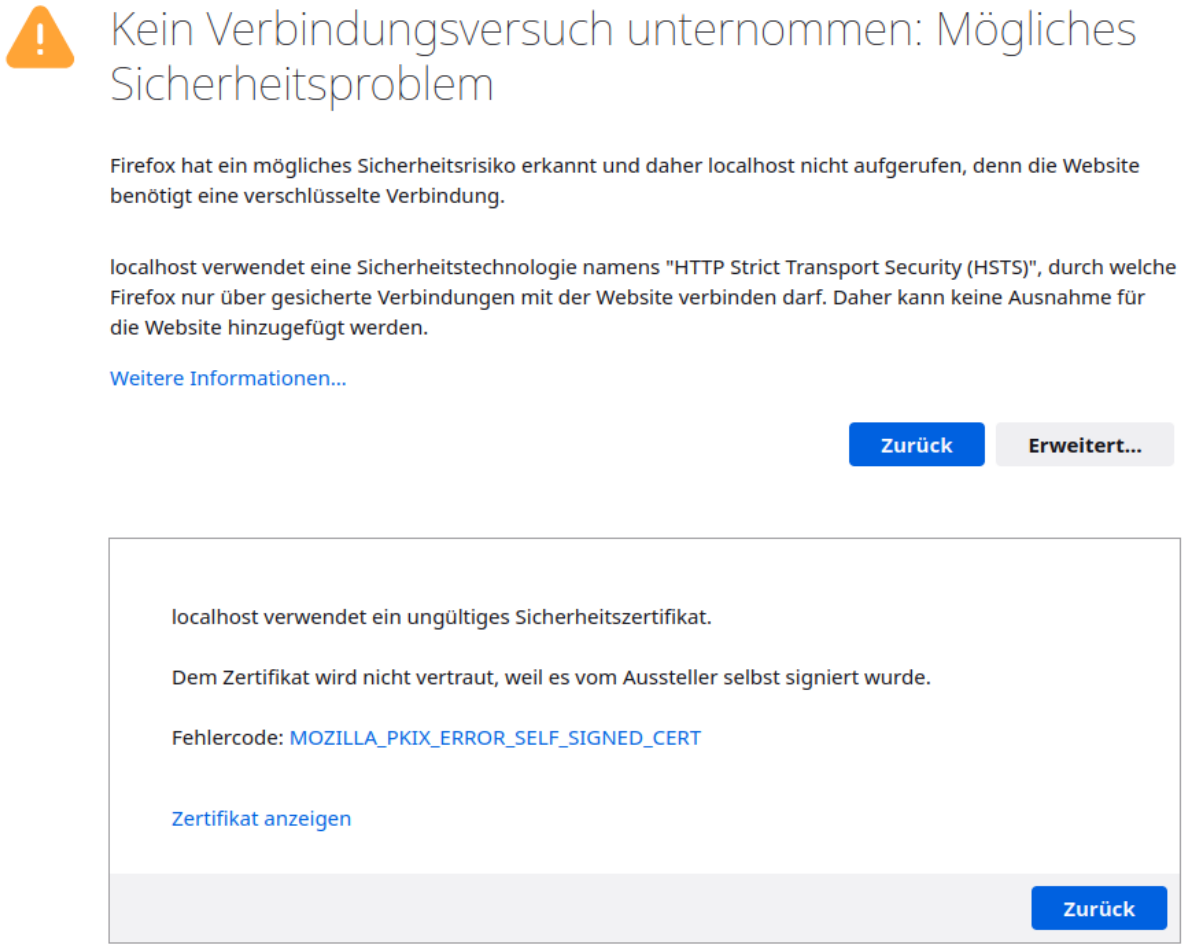
- **Erstellt ein Root Zertifikat**

```
openssl req -new -x509 -key ca.key -out ca.crt
```

- **Erstellt ein Zertifikat aus der Zertifikatsanfrage**

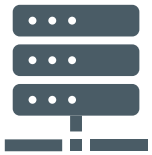
```
openssl x509 -req -in fk-vv.csr -CA ca.crt -CAkey
ca.key - CAcreateserial -out fk-vv.crt
```

Selbstsignierte Zertifikate - Probleme



Selbstsignierte Zertifikate - Probleme

Nicht verfügbare Funktionen



Service Worker

Push-Benachrichtigung

weitere

mkcert

- Tool zur Erstellung von lokal vertrauenswürdigen Entwicklungszertifikat
- Erstellt und installiert automatisch eine lokale CA im root-store des Systems

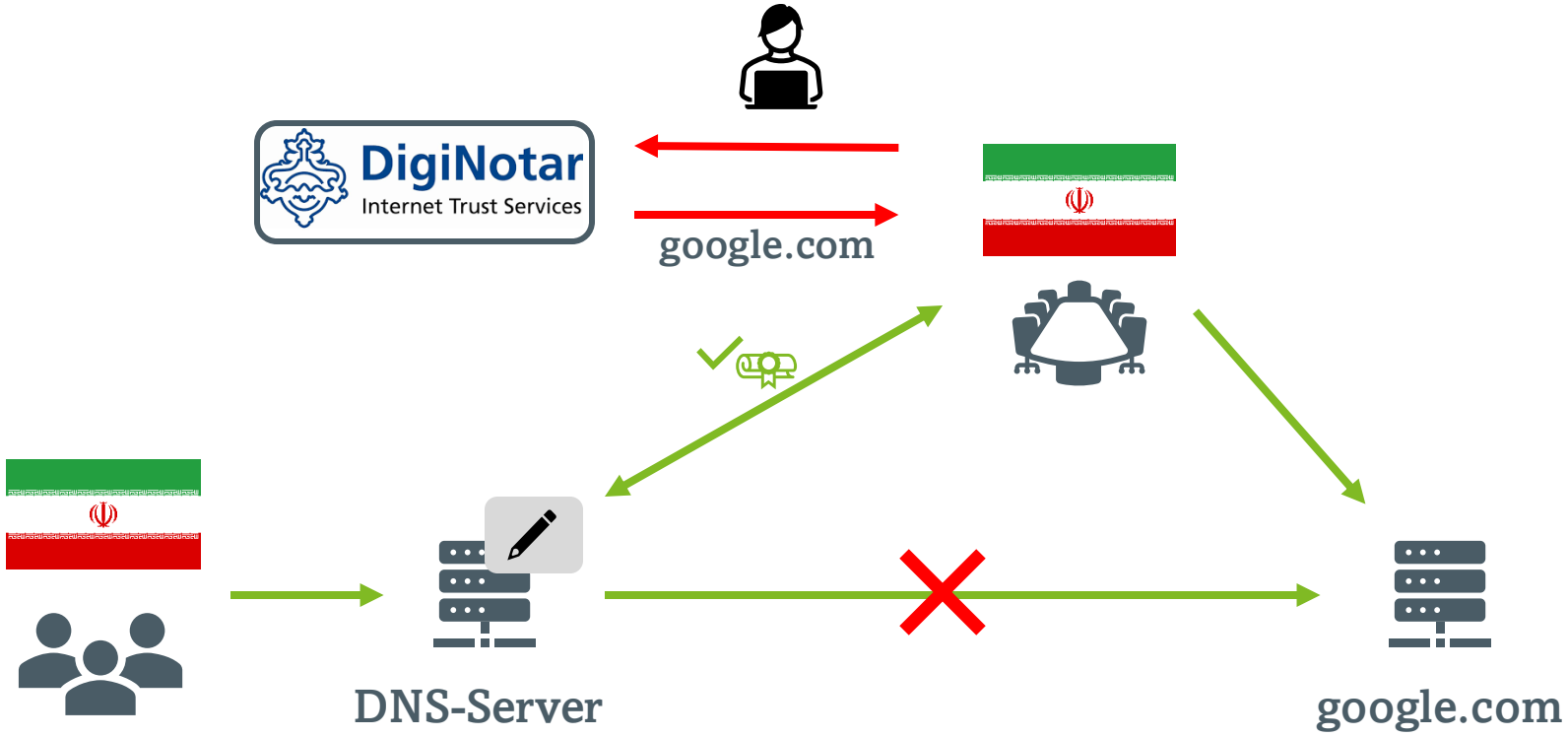
Die Datei `rootCA-key.pem`, die mkcert automatisch generiert, gibt einem die volle Macht, gesicherte Anfragen von dem Rechner abzufangen.
Datei nicht Teilen!

> https://mkcert.dev

Probleme mit CA‘s

- **Die letzte CA der Kette ist anfällig**
  - Man muss dieser vertrauen.
  - Wird diese gehackt ist die gesamte Internetsicherheit gefährdet.

Angriffe auf CA`s



Web of Trust (WoT)

- Ziel: Unabhängigkeit von CA's (auch wenn diese i.A. sicher sind)
- Eine Gruppe kann selbst festlegen, wem man vertraut.
- Gewissermaßen übernimmt die Gruppe die Aufgaben einer CA.
- Jeder Nutzer hat ein eigenes Schlüsselpaar und zertifiziert damit Schlüssel anderer Teilnehmer.
- nicht hierarchisch, sondern dezentral. Es ist eine eigene Trusted third party (TTP).

Web of Trust (WoT)

- **Vorteile**
  - Ausfallsicherheit
  - Jeder Benutzer kann festlegen, wie viel Vertrauen anderen Benutzern entgegen gebracht wird. Hierzu ist das Vorwissen und die Art des Kontaktes (vorort, telefonisch, etc.) entscheidend.
- **Nachteile**
  - Überprüfen der Identitäten, wann immer ein Zertifikat ausgestellt wird.
  - Mehrfaches zertifizieren lassen.