Digitale Zertifikate
Grundlagen und Anwendungen

PC-Treff-BB Aidlingen

Günter Waller

11.11.2017

Agenda

- Ziele der Informationssicherheit: C I A
- Grundlagen Verschlüsselung
- Asymmetrische Verschlüsselung
- RSA
- Hash, digitale Signatur
- PKI
- Worauf basiert Vertrauen? (Trust)
- Anwendungsfälle (Übersicht)
- Webserver
- Client Identifikation: SSO, SSH, VPN
- DNSSEC
- Qualifizierte elektronische Signatur
- Links

Ziele der Informationsicherheit: C I A

Zentrale Schutzziele der Informationssicherheit

- **#1 C = Confidentiality (Vertraulichkeit)**
- **#2 I = Integrity (Unversehrtheit [der Daten])**
- **A = Availability (Verfügbarkeit)**

Weitere Schutzziele

- **#3 Authenticity (Echtheit, Vertrauenswürdigkeit)**
- **#4 Non-repudiation (Verbindlichkeit)**
- **#5 Accountability (Zurechenbarkeit)**
- **Anonymität**

Mittel zur Zielerreichung #1-#5: Im Kern Kryptografie

Grundlagen Verschlüsselung

- Was brauche ich für eine Verschlüsselung?
  - Eine Abbildung (Funktion im mathematischen Sinne), umgesetzt i. d. R. In einem **Algorithmus**
    - Für alle gleich, idealerweise standardisiert, nicht geheim
    - Umkehrbar (Ver- und Entschlüsselung)
  - Ein Geheimnis (Secret), i.d.R. ein **Schlüssel**
    - Nur dem- oder denjenigen bekannt, welche die betroffenen Daten kennen sollen
- Problem Schlüsselaustausch
  - Annahme: Daten werden über unsicheren Kommunikationskanal ausgetauscht
    - Out-of-band Übermittlung des Schlüssels – oder ...

Asymmetrische Verschlüsselung

- Problem: Vermeidung der Übertragung des Schlüssels über das unsichere Medium
- Lösung: Es gibt 2 Schlüssel
  - Encryption Key $\mathcal{E}$ zur Verschlüsselung
  - Decryption Key $\mathcal{D}$ zur Entschlüsselung
- $\mathcal{E}$ und $\mathcal{D}$ bilden ein Paar
- $\mathcal{E}$ muß nicht geheim gehalten werden - im Gegenteil.
- $\mathcal{D}$ muß geheim bleiben, gehört nur einer Person.
- $\mathcal{D}$ darf nicht leicht aus $\mathcal{E}$ herzuleiten sein.
- Die Verschlüsselungsfunktion ist eine Einwegfunktion (one-way function): Eine Richtung geht leicht, die andere schwer.
- Hat man eine Zusatzinformation ($\mathcal{D}$), geht es leicht. Dann spricht man von einer Falltürfunktion (trap door function).

Beispiel: RSA

- Geschichte: Anfang der 70er Jahre wurde beim britischen GHCQ ein asymmetrisches Verfahren entwickelt, aber nie veröffentlicht.
- 1976: Diffie und Hellman (Stanford University) veröffentlichen ein theoretisches Paper über Public Key Kryptographie. Zu dem Zeitpunkt betrachten sie das Problem als ungelöst.
- 1977: Rivest, Shamir und Adleman (MIT) stießen beim Versuch, obiges Papier zu widerlegen auf ein funktionierendes Verfahren: RSA. Der Artikel zeigt auch schon auf, dass das Verfahren sowohl für Signaturen als auch für Vertraulichkeit (Verschlüsselung) geeignet ist.
- Daraus wird 1983 ein Patent und es entsteht eine Firma für Security Produkte. Das Patent ist allerdings seit 2000 abgelaufen.

RSA Algorithmus 1/3

- Der öffentliche Schlüssel besteht aus einem Paar ganzer Zahlen (e,n).
- Zu verschlüsselnde Nachricht M (Message) durch eine Zahl repräsentieren (<n). Kodierung ist egal. Längere Nachricht in mehrere kleine zerlegen.
- Mit der Verschlüsselungsfunktion E (Encrypt) ermittelt man nun die verschlüsselte Nachricht C (Ciphertext):

$$C = E(M) = M^e \pmod n$$

- Das Gegenstück, der private Schlüssel, besteht aus einem weiteren Paar (d,n) mit dem gleichen n.
- Dazu gehört die Entschlüsselungsfunktion D (Decrypt):

$$M = D(C) = C^d \pmod n$$

- Wichtig: Die Länge von C wird durch E nicht größer als n-1.

RSA Algorithmus 2/3

- Bis dahin klingt es einfach. Der Aufwand steckt

a) im Finden zweier großer Primzahlen p und q als Startwert für die Schlüsselerzeugung [1]

b) in der Erzeugung der Schlüssel:

   i. Beschreibung in Wikipedia

   ii. Mathematischer Beweis im RSA-Originalartikel

c) in der Schlüssellänge, d.h. im erforderlichen Rechenaufwand

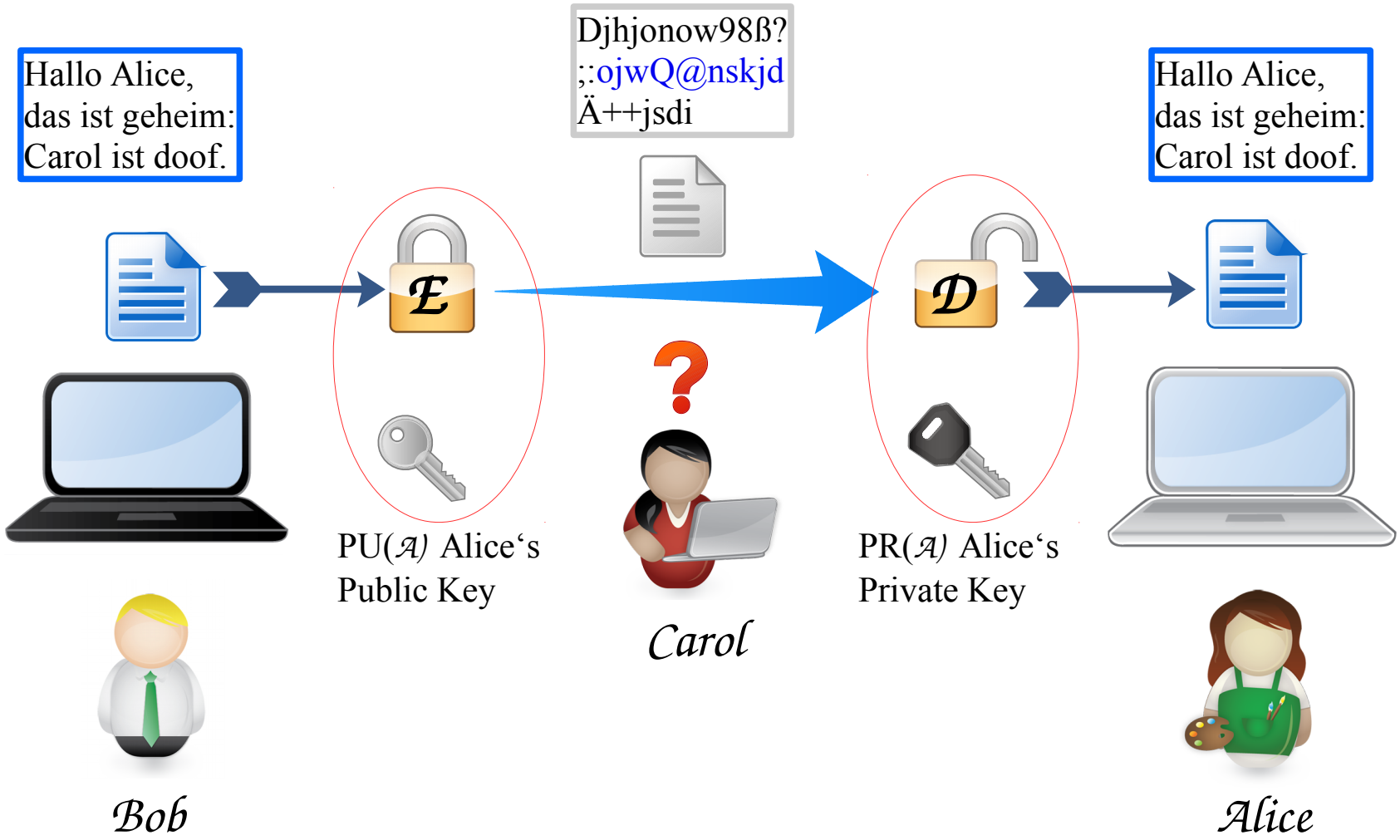
---

[1] Es gilt n = p x q. Man erzeugt Zufallszahlen in der gewünschten Länge und testet, ob sie prim sind. Im Mittel braucht man dafür 115 Versuche. Wir verwenden Zahlen, die zu groß sind, um das mit Brute Force durchzuprobieren. Es gibt keinen Beweis, sondern nur Tests, die teilbare Zahlen erkennen. Davon werden sehr viele ausgeführt, die alle negativ ausgehen müssen. Es bleibt ein geringes, vernachlässigbares Restrisiko, dass man eine nicht-Primzahl erwischt hat. Dann wird D(E(M)) ≠ M.

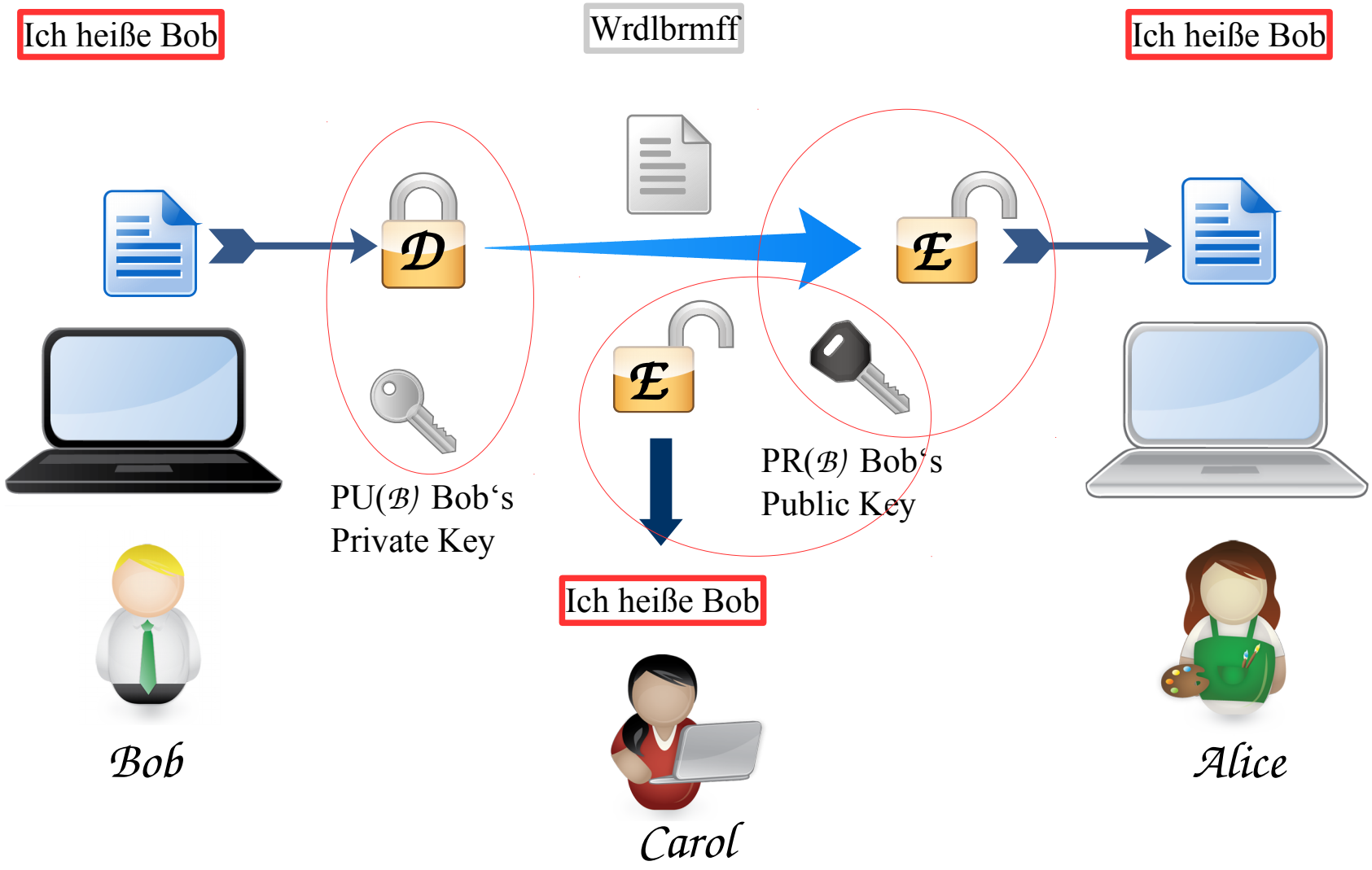
RSA Algorithmus 3/3

- Beschleunigung der Rechnung mit dem Chinesischen Restsatz (CRT-RSA)
- Das unmodifizierte RSA-Verfahren ist angreifbar, weil es deterministisch ist, d.h. der gleiche Klartext gibt immer das gleiche Chiffrat. Damit sind Angriffe möglich, ohne den Schlüssel zu faktorisieren. Lösung: Der Klartext wird mit einem Zufallswert ergänzt (Padding) und dann verschlüsselt.
- Aus Effizienzgründen wird RSA in der Regel in Hybridverfahren mit symmetrischen Verfahren kombiniert. Es wird zufällig ein Sitzungsschlüssel für ein symmetrisches Verfahren erzeugt, der per RSA verschlüsselt und zusammen mit der Nachricht übertragen wird.
- Für die Sicherheit von RSA sind Primzahlen mit mehreren hundert Dezimalstellen (mindestens 2048 Bit) erforderlich. Damit können symmetrische Schlüssel jeder üblichen Länge verschlüsselt werden (z.B. AES mit einer Schlüssellänge von 128, 168 oder max. 256 Bit)
- Further Reading: http://www.muppetlabs.com/~breadbox/txt/rsa.html

Geheime Daten, verschlüsselt übertragen über ein unsicheres Netz



Öffentliche Daten, vom Absender signiert, Sender-Identität und Integrität bewiesen



Hash – was ist das?

Eine Hash-Funktion bildet Daten beliebiger Länge auf Daten einer festen Länge ab.

- Eigenschaften einer kryptografischen Hash-Funktion:
  - Deterministisch
  - Leicht zu berechnen
  - Praktisch nicht umkehrbar
  - Kleine Änderungen am Input bewirken einen völlig anderen Output
  - Praktisch unmöglich, ein zweites Input-Datum zu finden, das den gleichen Output erzeugt
- Anwendungen in kryptografischen Verfahren, insbes. als **digitale Signaturen** und für Authentifizierung (Password Hashing). Nicht-kryptografische Anwendungen: Hash-Tables, digitale Fingerabdrücke, Prüfsummen (bei Downloads).

Hash – Praxis

- Die älteren Hash-Verfahren (MD4, MD5, SHA-0, u.a.) gelten inzwischen als unsicher. Gegen SHA-1 wurde 2017 eine Kollision berechnet.
- Aktuell sind:
  - SHA2: Oberbegriff für SHA-224, SHA-256, SHA-384 und SHA-512. Die Zahl steht für die Länge des Hashwertes.
  - SHA3: Zukunftssicherer als SHA-2, verwendet andere Technik. Soll bereitstehen, wenn eines Tages SHA-2 gebrochen wird.
- Verschärfungen bei Passworten (zur Erschwerung von Brute-Force-Angriffen mit Rainbow Tables von bekannten Hashwerten):
  - Salt: Klarnachricht wird um einen Zufallswert erweitert, dann wird Hash gebildet. PW geht nicht im Klartext durchs Netz.
  - Pepper: Server ergänzt Klartext um einen geheimen Wert, der nicht zusammen mit den Hashes gespeichert wird.
  - Grundsätzlich sollten PW **nie** im Klartext gespeichert werden.

Prinzip der digitalen Signatur

Hier werden Hash- und Verschlüsselungs-Funktion zu einem Verfahren zusammengefügt.

Siehe z.B.: http://www.itwissen.info/Digitale-Signatur-digital-signature-DSig.html

PKI – Was ist das? Wozu dient das?

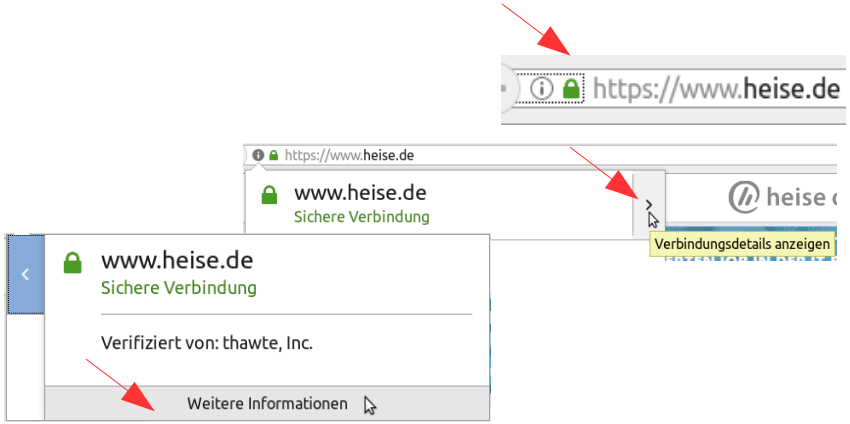
PKI = Public Key Infrastructure

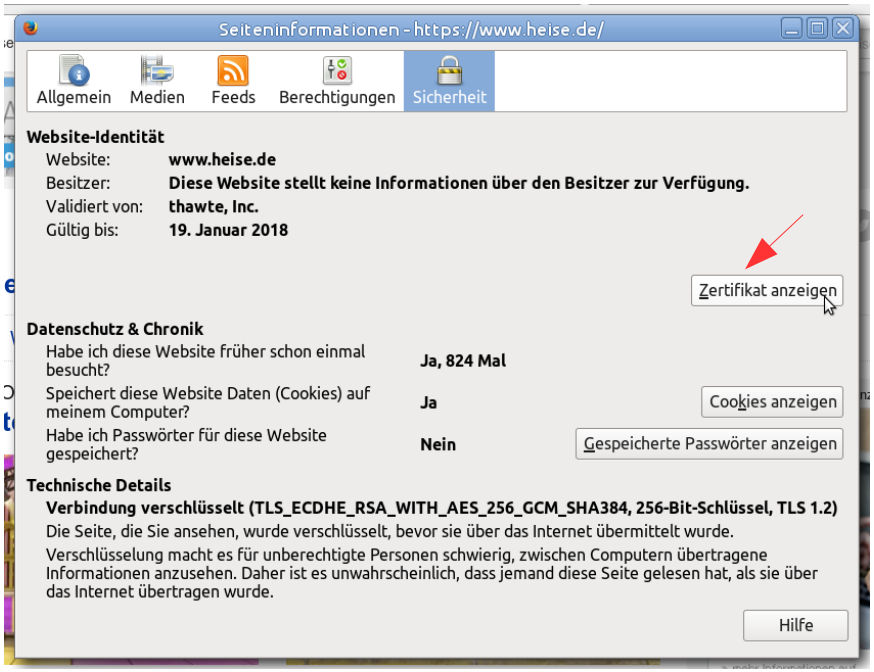
- Ziel (allgemein): **Übermittlung des öffentlichen Schlüssels** (public key) **in asymmetrischen Kryptosystemen**, Garantie der Echtheit bzw. der **Korrektheit der zugehörigen Identität**.
- Mittel zum Zweck: **Digitales Zertifikat**. Signiert vom **Aussteller**.
  - Authentizität der Signatur durch Überprüfung des Zertifikates des Ausstellers.
  - usw. usw.
  - Am Ende (oder Beginn) dieser Kette (**Trust** Chain) muss ein Zertifikat stehen, dem man auf andere Art und Weise, also ohne weiteres Zertifikat vertraut: das Root-Zertifikat.
    - Selbst ausgestellt, selbst als vertrauenswürdig importiert
    - Vom Browserhersteller vorinstalliert
    - Vom Betriebssystem vorinstalliert

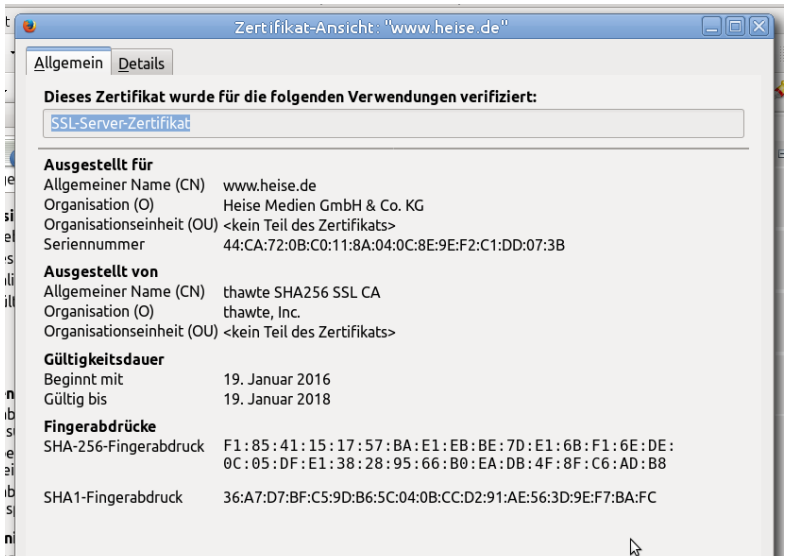
Demo

- https://www.heise.de/

1. Im Browser auf das Schloss klicken
   a) > Verbindungsdetails
   b) Weitere Informationen
2. Zertifikat anzeigen





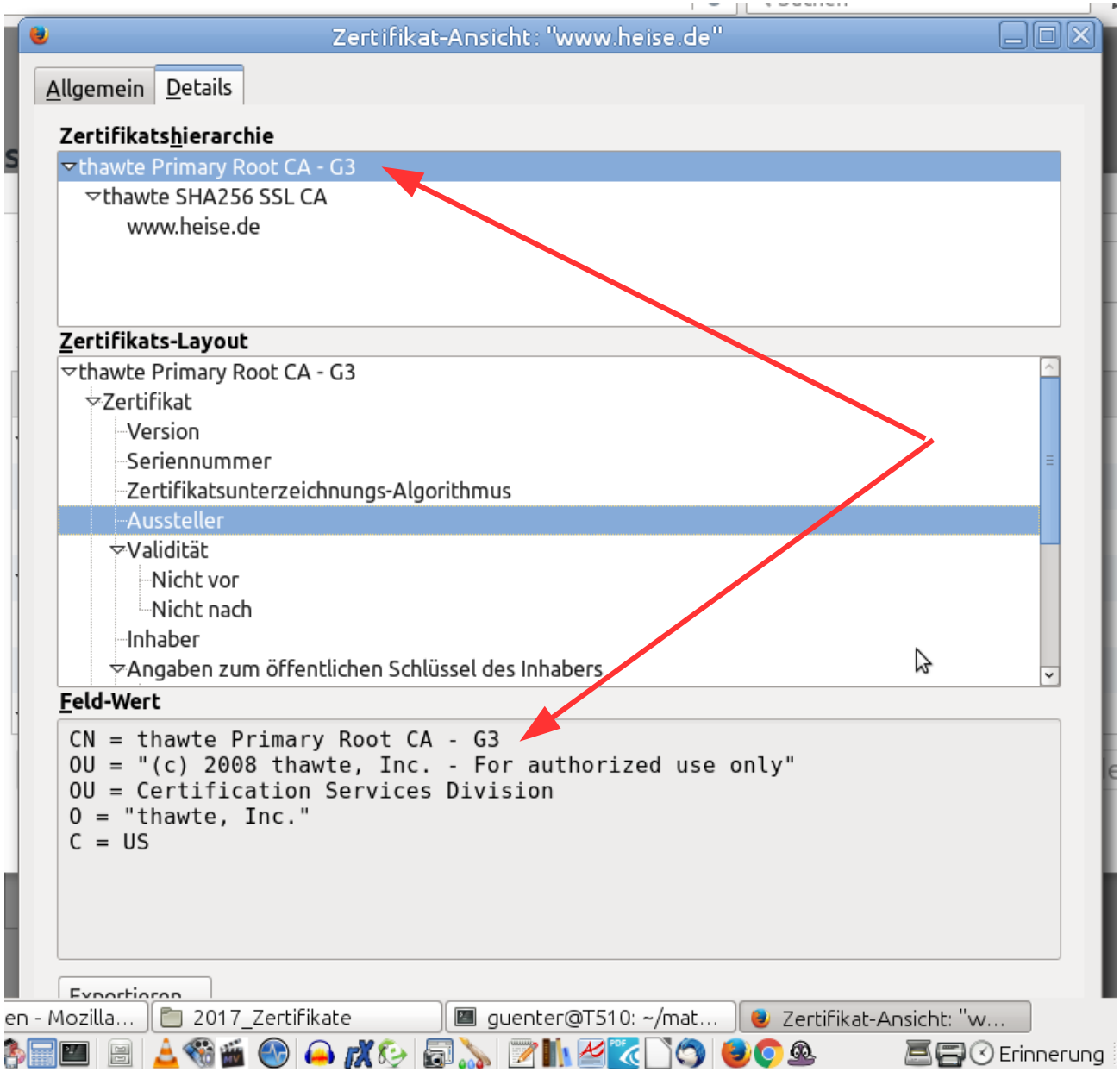


Demo (Forts.)

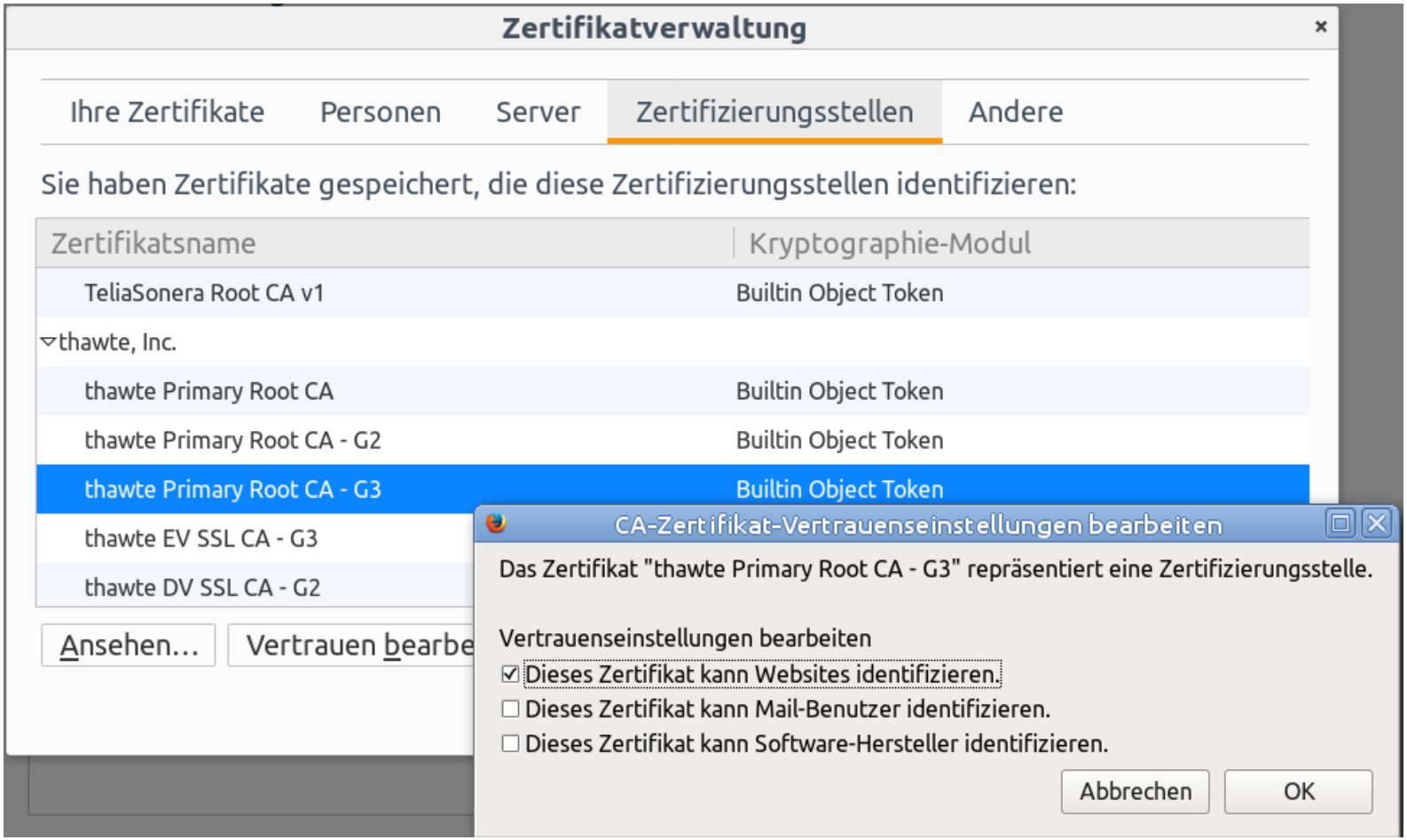
- Walkthrough durch die Details:
  - Zertifikatshierarchie
    - Root-Zertifikat ist selbstsigniert
  - Version (X.509 V3), Seriennummer, Signaturalgorithmus (PKCS), Validität, Inhaber (LDAP-Attribute)
  - Erweiterungen: CRL-Verteilungspunkte (Download CRL), Zertifizierungsstellen-Informations-Zugriff (u.a. OCSP-Link)
  - CRL ansehen mit openssl:

```
openssl crl -inform DER -noout -text -in tg.crl > crl_view.txt
```

Root-Zertifikat ist selbstsigniert

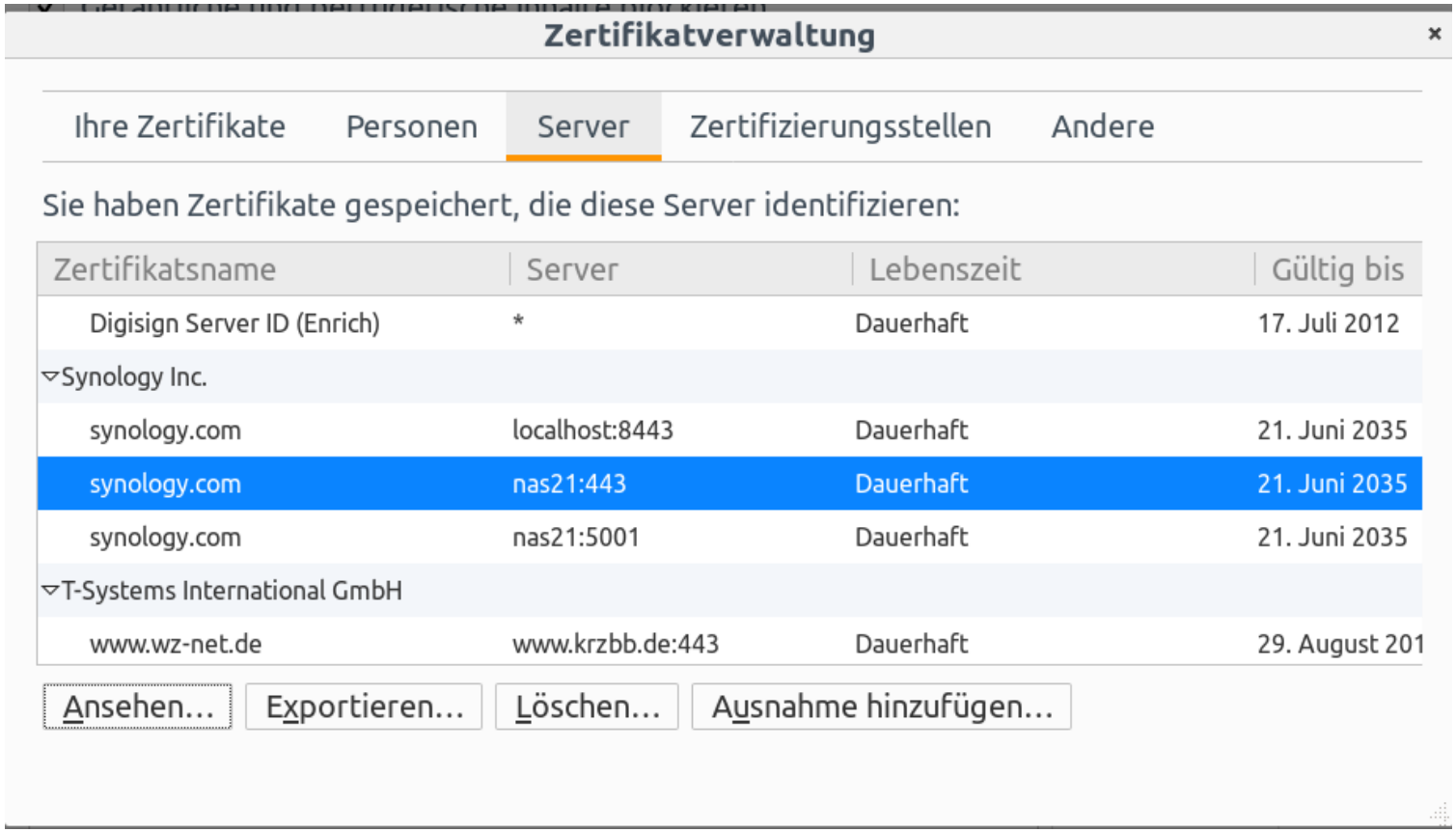


Trust für Root-Zertifikat kommt vom Browser



Ausnahmen im Browser hinzufügen

Für eigenes NAS Ausnahme(n) hinzugefügt.



Bestandteile einer PKI

- **Digitale Zertifikate**: Digital signierte elektronische Daten, die sich zum Nachweis der Echtheit von Objekten verwenden lassen (Definition aus Wikipedia). Enthält Verfallsdatum.
- Zertifizierungsstelle (CA) stellt das Zertifikat aus.
- Registrierungsstelle (RA) bekommt Antrag, prüft die Richtigkeit der Angaben (Beispiele: siehe Let‘s Encrypt, oder Überprüfung Ausweis), informiert CA.
- Zertifikatssperrliste (CRL) enthält die Zertifikate, die vor Ablauf ihrer Gültigkeit widerrufen werden. Client (z.B. Browser, sollte dies prüfen).
- Verzeichnisdienst (LDAP Server) für Zertifikate und CRLs.
- Validierungsdienst (VA) führt Gültigkeitsprüfungen in Echtzeit durch (z.B. OCSP), falls die Anwendung dies erfordert.
- Subscriber (Zertifikatsinhaber), Participant (Nutzer, der vertraut)

Dateiformate für Zertifikate

- Laut Wikipedia:

.CER – DER- oder Base64-kodiertes Zertifikat

.CRT – DER- oder Base64-kodiertes Zertifikat

.CSR – Base64-kodierte Zertifizierungsanfrage des öffentlichen Schlüssels (plus weitere Metadaten des Besitzers) an eine CA, umschlossen von „-----BEGIN CERTIFICATE REQUEST-----“ und „-----END CERTIFICATE REQUEST-----“

.DER – DER-kodiertes Zertifikat

.P12 – PKCS#12, kann öffentliche Zertifikate und private Schlüssel (Kennwort-geschützt) enthalten.

.P7B – Siehe .p7c

.P7C – PKCS#7-signierte Datenstruktur ohne Dateninhalt, nur mit Zertifikat(en) oder Sperrliste(n)

.PEM – Base64-kodiertes Zertifikat, umschlossen von „-----BEGIN CERTIFICATE-----“ und „-----END CERTIFICATE-----“

.PFX – Siehe .p12

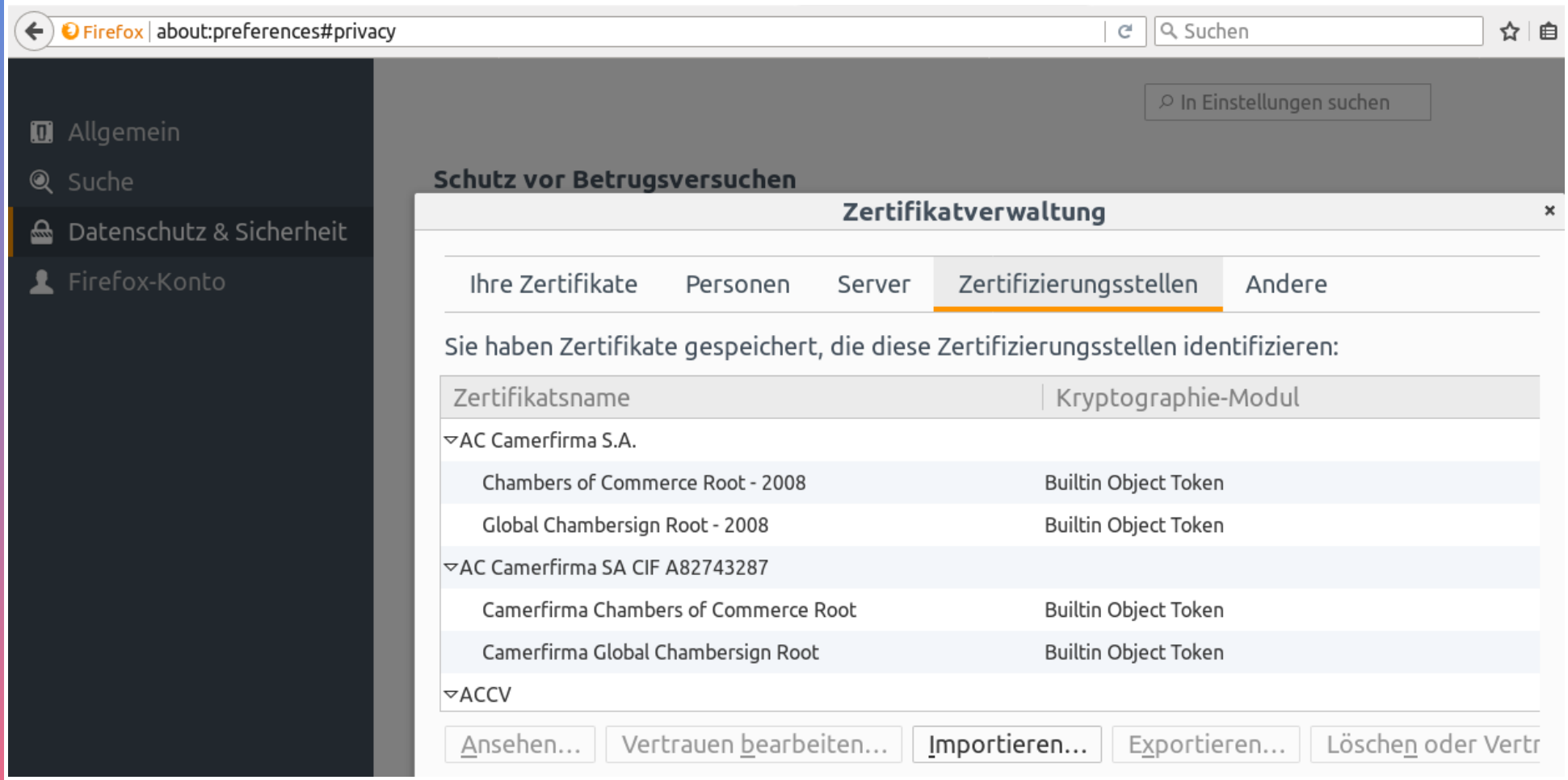
- DER = Distinguished Encoding Rules
- CSR = Certificate Signing Request

Worauf basiert Vertrauen (Trust)?

Bei dem beschriebenen System handelt es sich um ein hierarchisches Gebilde. Das Vertrauen steht und fällt damit, dass Vertrauen in die Ur-CA am Anfang der Kette besteht, der Root-CA.
Laut Wikipedia wird dieses Feld von einer kleinen Zahl von Unternehmen dominiert, genannt werden Symantec (ex: VeriSign), Comodo, GoDaddy (Webhoster), die 3/4 des Marktes für TLS-Zertifikate von Webservern halten sollen.
In der Vergangenheit gab es gelegentlich Zwischenfälle, bei denen eine Root-CA kompromittiert wurde. Als Folge mußten die entsprechenden Zertifikate aus den Browsern, Betriebssystemen entfernt werden, z.T. auch händisch.

- Diginotar (2011)
- Google und Symantec (aktuell)

Beispiel: Firefox Zertifikatverwaltung



Anwendungsfälle

- https://de.wikipedia.org/wiki/RSA-Kryptosystem#Anwendungsgebiete
- Internet- und Telefonie-Infrastruktur: **X.509-Zertifikate**
- Übertragungs-Protokolle: **IPsec**, **TLS**, **SSH**, WASTE
- E-Mail-Verschlüsselung: **OpenPGP**, **S/MIME**
- Authentifizierung französischer Telefonkarten
- Kartenzahlung: EMV
- RFID Chip auf dem deutschen Reisepass
- Electronic Banking: HBCI
- Single Sign-On (SAML, OpenId, OAuth)
- **VPN**
- **DNSSEC**

„Schlüsselverwendung“ - öffentlicher Schlüssel

- **Digitale Signatur**: für digitale Signaturen, nicht für Nichtabstreitbarkeit, d.h. eher kurzfristiger Charakter.
- **Nichtabstreitbarkeit**: für digitale Signaturen eines Nichtabstreitbarkeitsservice, z.B. Notariatsservice.
- **Schlüsselverschlüsselung**: für die Verschlüsselung von anderen Schlüsseln oder Sicherheitsinformationen.
- **Datenverschlüsselung**: zur Verschlüsselung von Benutzerdaten (außer andere Schlüssel).
- **Schlüsselvereinbarung**: Diffie Hellman Algorithmus soll für die Schlüsselvereinbarung verwendet werden.
- **Zertifikatsignatur**: für die Verifikation von Signaturen auf Zertifikaten, d.h. sinnvoll bei CA-Zertifikaten.
- **CRL-Signatur**: für die Verifikation von Signaturen auf CRLs, sinnvoll bei CA-Zertifikaten.
- **Nur Verschlüsselung/nur Entschlüsselung**: nur mit Schlüsselvereinbarung nach Diffie Hellman sinnvoll.

Erweiterte Schlüsselverwendung

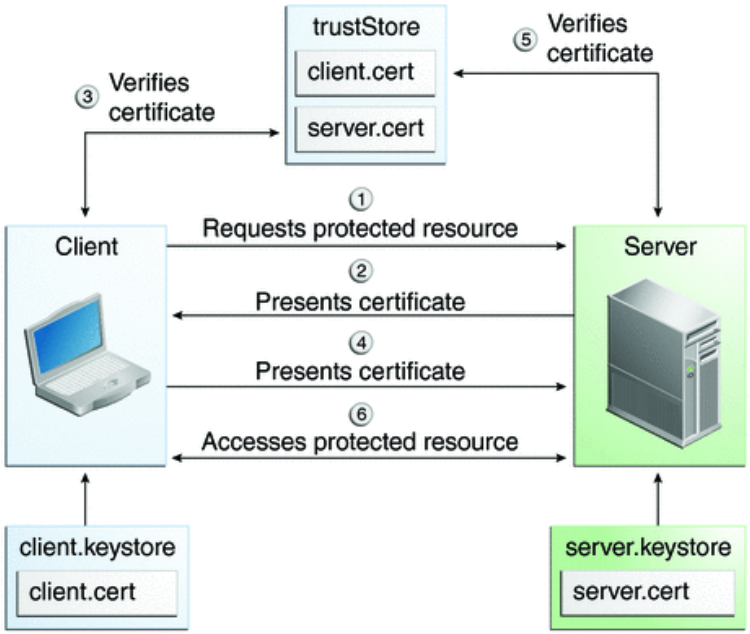
In der Abteilung „Extended Key Usage“ des Zertifikates finden sich wichtige Punkte (hier: von Microsoft definiert):
- Serverauthentifizierung (Webserver)
- Clientauthentifizierung

u.a.

Quelle (z.B.): https://www.cryptas.com

Webserver

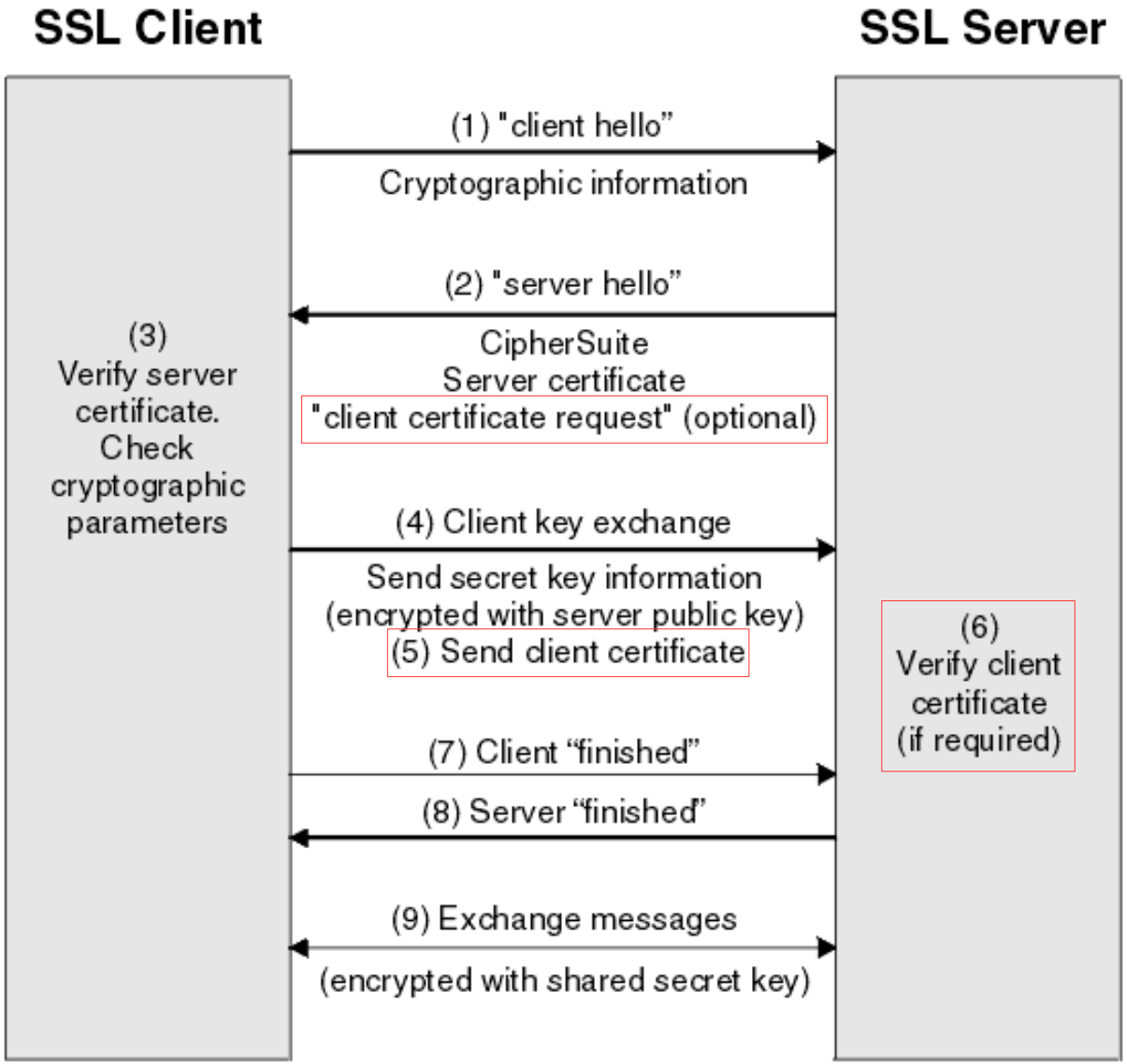
- Der Server identifiziert sich bei HTTPS mit einem Serverzertifikat. Bei HTTP gibt es keine Gewähr, dass es sich nicht um einen Fake Server handelt.
- Auch der Client kann sich gegenüber dem Server mit einem Zertifikat identifizieren, d.h. in dem Fall authentifizieren. Das Zertifikat enthält dann in Attributen die Identität. Der Server validiert das Zertifikat und prüft die Identität gegen sein Benutzerverzeichnis. Es fließt kein Passwort. Gleichzeitig dient das Zertifikat dem TLS-Verbindungsaufbau.



TLS Handshake

- Verständigung auf das verwendete Protokoll
- Auswahl Krypto-Algorithmus
- Austausch und Validierung der Zertifikate
- Generierung gemeinsamer Schlüssel
- Nutzdaten symmetrisch verschlüsselt

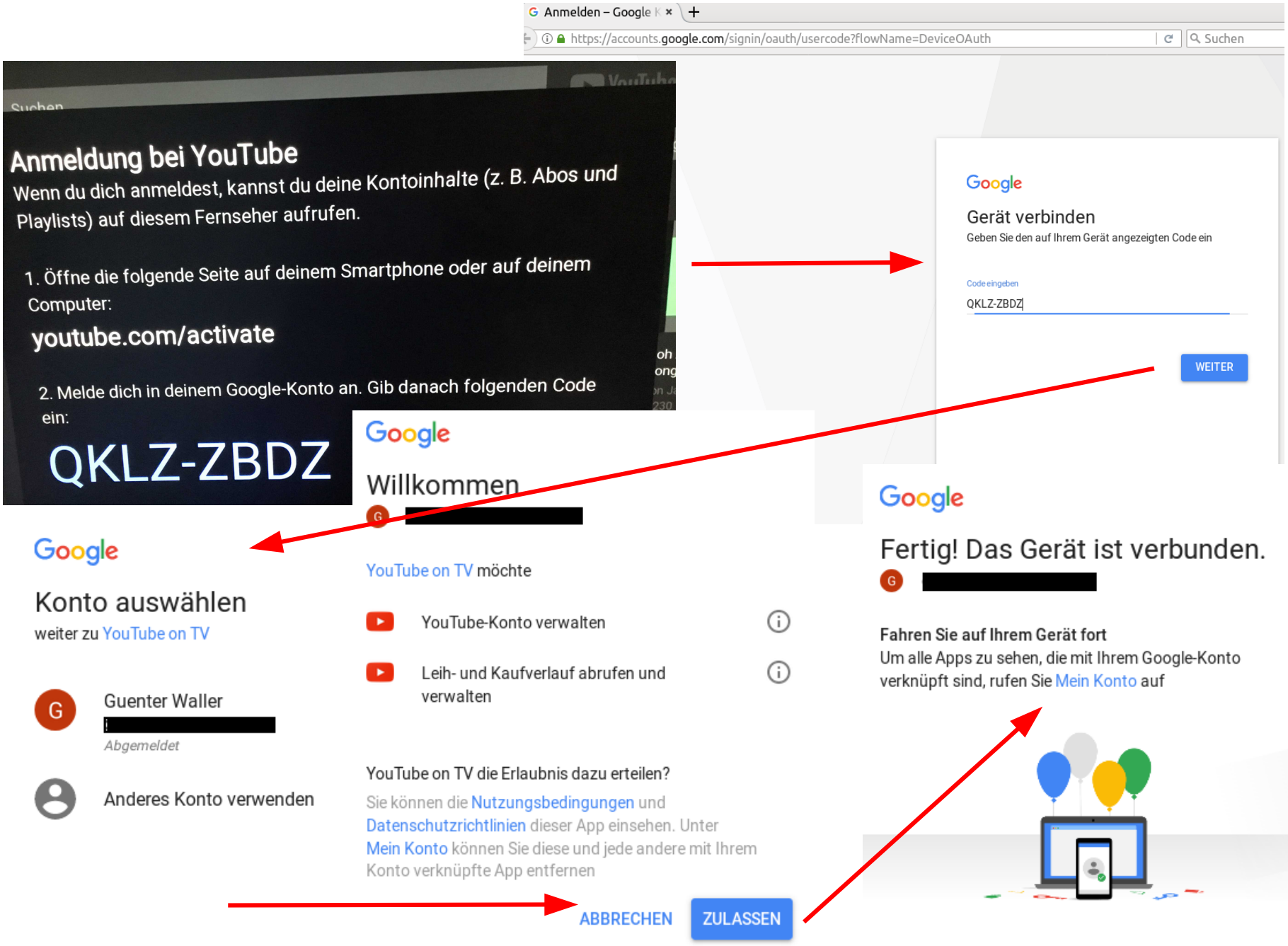
TLS Handshake mit Client Zertifikat



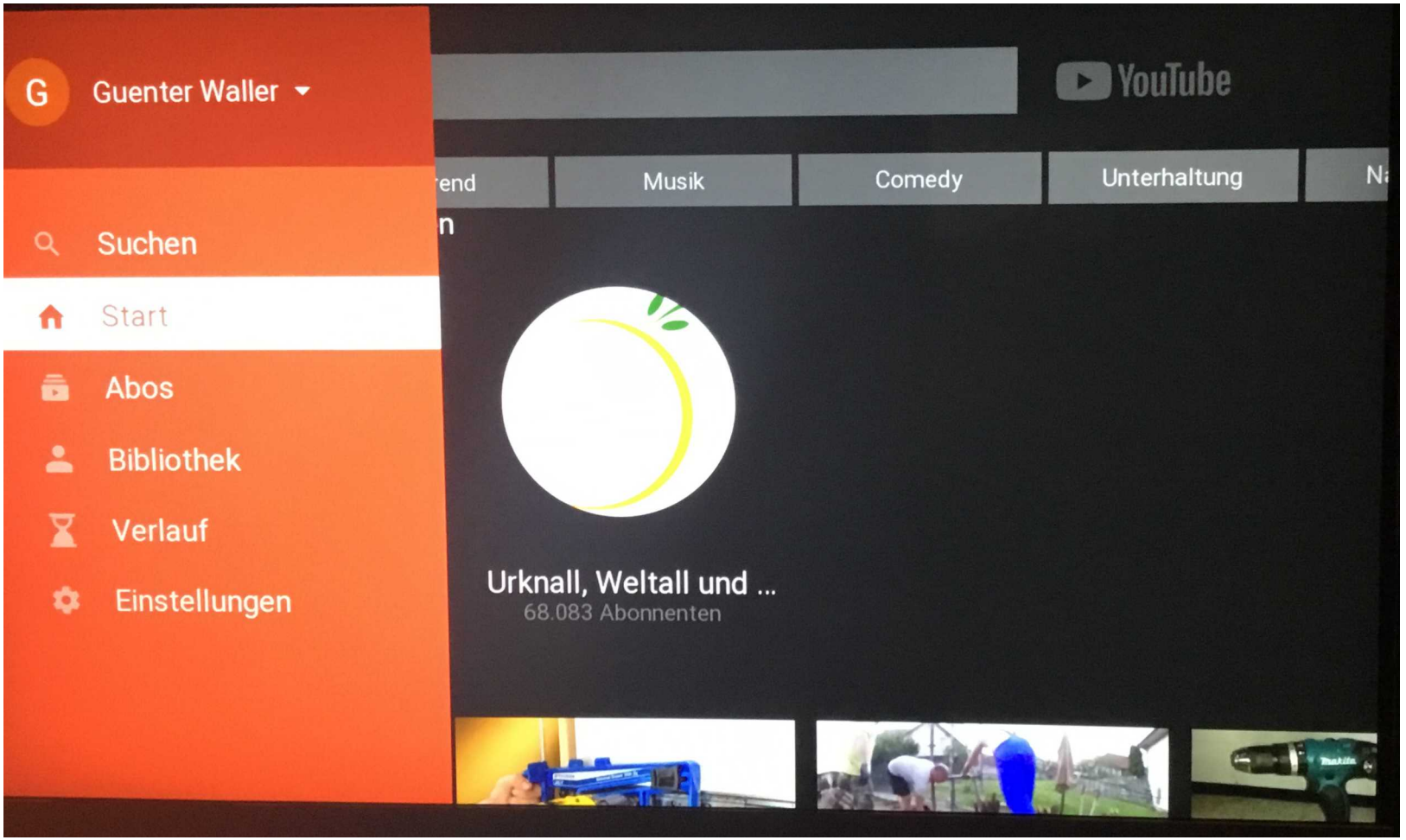
Single Sign-On

- **SAML**, ein auf XML-formatierten Tokens aufbauendes SSO-Verfahren, enthält digitale Zertifikate als Bestandteil.
- **OAuth**, ein Autorisierungsprotokoll zwischen Client, User (Resourceeigner) und Resourceserver erlaubt dem User, an den Client Rechte zu vergeben, ohne diesem das eigene passwort am Server geben zu müssen. Das Mittel ist ein Access Token, der zeitlich und funktional limitiert ist.
  - Mein Beispiel: YouTube App auf dem Smart TV verlangt von mir, mich auf einem anderen Gerät bei Google anzumelden und dort den angezeigten Code einzugeben, so dass die App unter meiner Id agieren kann.
  - Für Banking wird jetzt OAuth mit Client-Zertifikaten verknüpft, um es wasserdicht zu machen.

OAuth am TV



OAuth am TV



SSH

- Rückgriff auf OpenWRT Vortrag von 2013
  - Inzwischen wurde allerdings der SSH Service Dropbear durch OpenSSH ersetzt, daher gibt es serverseitig leichte Abwandlungen:
    - `/root/-ssh/authorized_keys` (neuer Pfad)
    - Keytype ssh-dss nicht mehr unterstützt
  - Auf dem Linux Client deshalb ssh-rsa erzeugt:
    - `ssh-keygen -t rsa`
    - Eintrag in `~/.ssh/config`
      - `PubkeyAcceptedKeyTypes ssh-rsa`

SSH mit Zertifikat

- Schritte pro Client:
  - Erzeugen Public/Private Key Pair
    - Dabei Passphrase für spätere Zugriffe festlegen
  - Übertragen des Public Key an den Router
  - Auf dem Router den Key als berechtigten Client-Key eintragen
  - Testen – beim Login wird (grafisch) nach der vorher festgelegten Passphrase gefragt, es geht kein Passwort über die Leitung.
- Android Client: App „VX ConnectBot“
- IOS: App „Termius“

SSH mit Zertifikat (Linux Client)

```
guenter@T42p:~$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/guenter/.ssh/id_dsa):
Enter passphrase (empty for no passphrase): <Eingabe der Passphrase>
Enter same passphrase again: <Eingabe der Passphrase>
Your identification has been saved in /home/guenter/.ssh/id_dsa.
Your public key has been saved in /home/guenter/.ssh/id_dsa.pub.
The key fingerprint is:
xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx guenter@T42p
The key's randomart image is:
+--[ DSA 1024]----+
|     *o  .o++.  + |
|  =     . o...+   |
|   D  =  ..o o    |
|     .   .  D     |
|    oSo =         |
|     * o          |
|      .           |
|                  |
|                  |
+------------------+
```

```
guenter@T42p:~$ ssh -p 4711 root@192.168.1.1
<... grafischer lokaler Prompt ...>
 BusyBox v1.15.3 (2011-11-24 00:44:20 CET) built-in shell (ash)
Enter 'help' for a list of built-in commands.

  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 Backfire (10.03.1, r29592) ------------------------
  * 1/3 shot Kahlua     in a shot glass, layer Kahlua
  * 1/3 shot Bailey's   on the bottom, then Bailey's,
```

```
guenter@T42p:~$ scp -P 4711 ~/.ssh/id_dsa.pub root@192.168.1.1:/tmp
```

```
root@openwrt:~# cd /etc/dropbear/
root@openwrt:/etc/dropbear# ls
dropbear_dss_host_key  dropbear_rsa_host_key
root@openwrt:/etc/dropbear# cat /tmp/id_dsa.pub >> authorized_keys
root@openwrt:/etc/dropbear# chmod 0600 authorized_keys
```

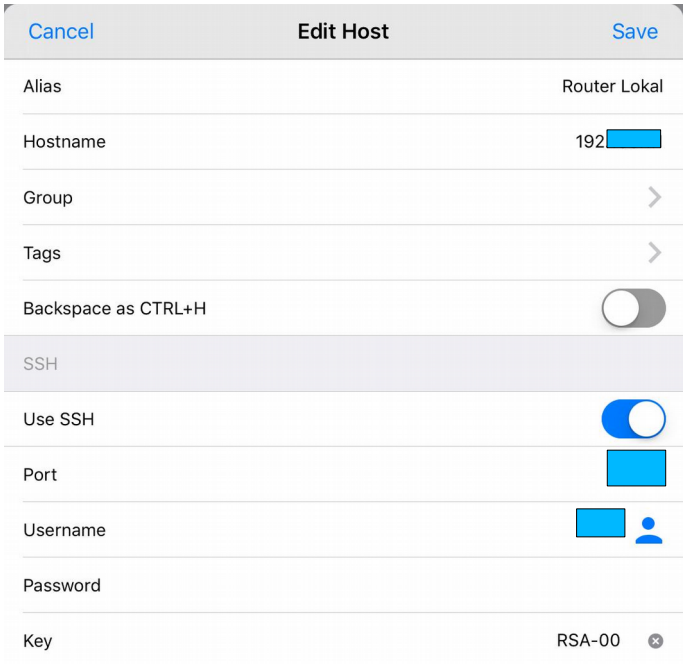
PC-Treff-BB Aidlingen

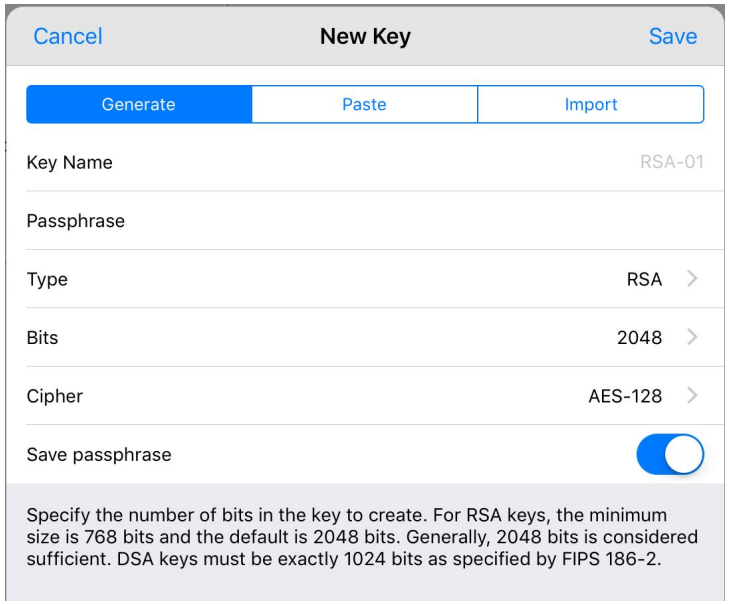
OpenWRT

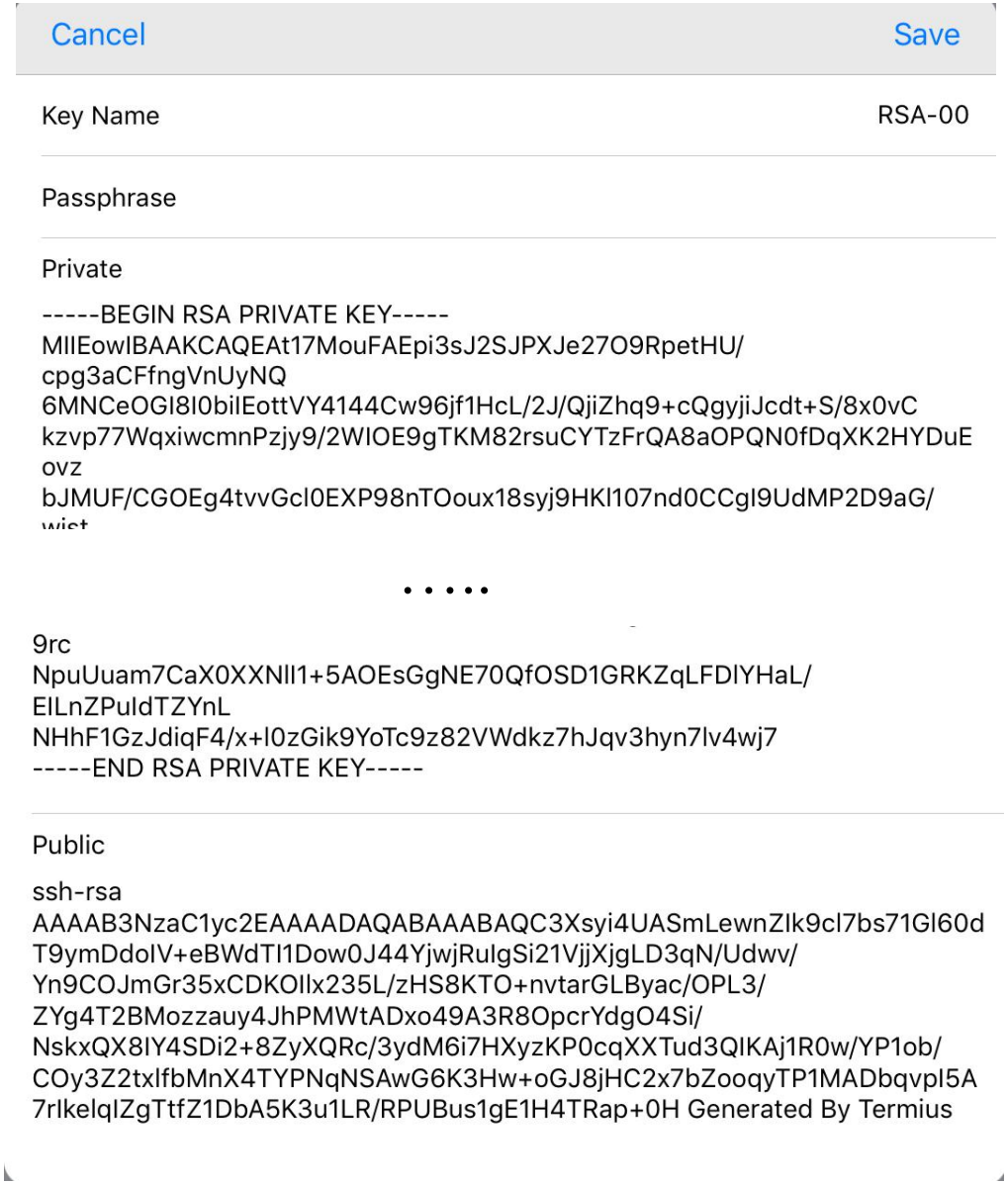
PC-Treff-BB Aidlingen

OpenWRT

IOS App „Termius“







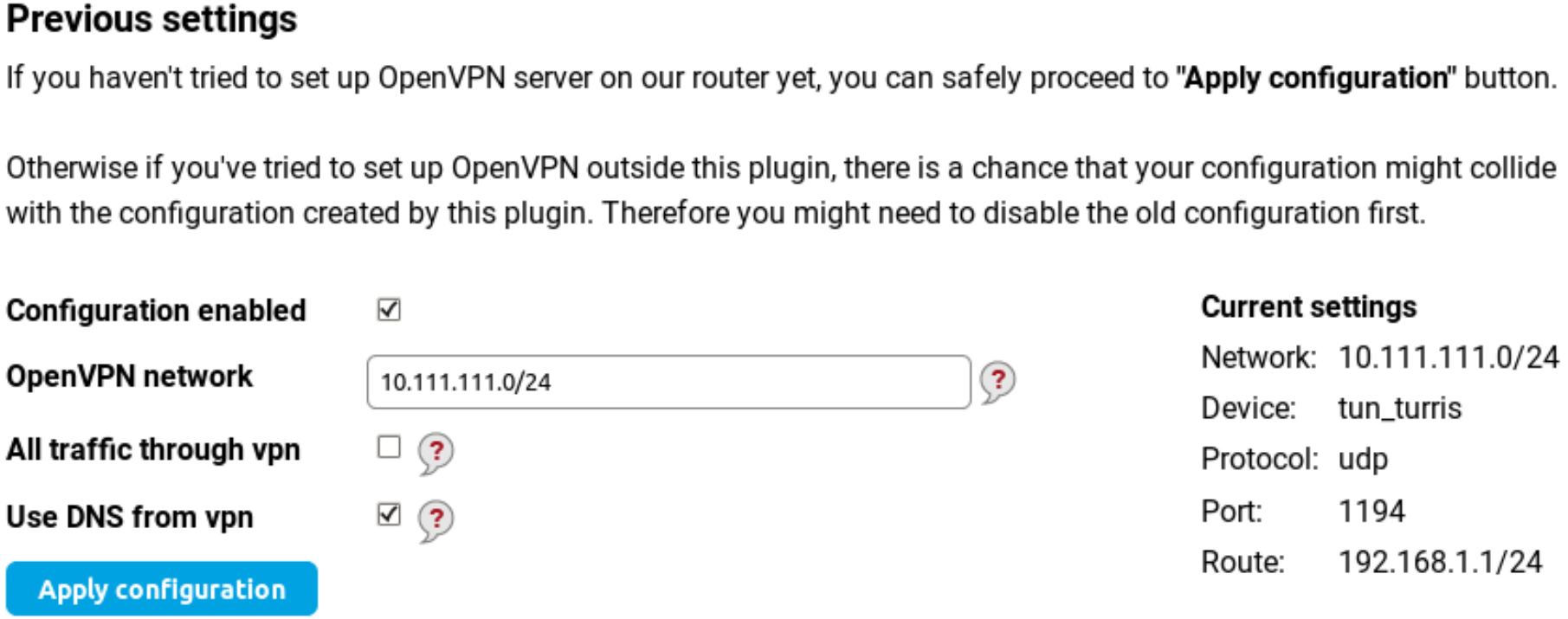
OpenVPN (1/4)

- OpenVPN verwendet zertifikatsbasierte Authentifizierung
- Mein OpenWRT-Router (Turris Omnia) hat eine vereinfachte Konfiguration über die Web-Oberfläche
  - Vereinfachtes Installieren OpenVPN Paket
  - CA generieren (dauert etwas). Ergebnis im Verzeichnis `/etc/ssl/ca/openvpn`

```
-rw-r--r--    1 root     root          6704 Sep  3 00:32 01.crt
-rw-r--r--    1 root     root          1582 Sep  3 00:32 01.csr
-r--------    1 root     root          3268 Sep  3 00:32 01.key
-rw-r--r--    1 root     root          6704 Sep  3 00:32 01.pem
-rw-r--r--    1 root     root          1862 Sep  3 00:32 ca.crt
-r--------    1 root     root          3272 Sep  3 00:32 ca.key
```

OpenVPN (2/4)

Es wird dann auf der gleichen Seite eine Konfiguration vorgeschlagen. Ich habe sie so gelassen und „applied“. Dabei geht es um die Networking-Aspekte.



OpenVPN (3/4)

In der Konfigurationsdatei finden sich folgende, mit Zertifikaten befassten Zeilen:

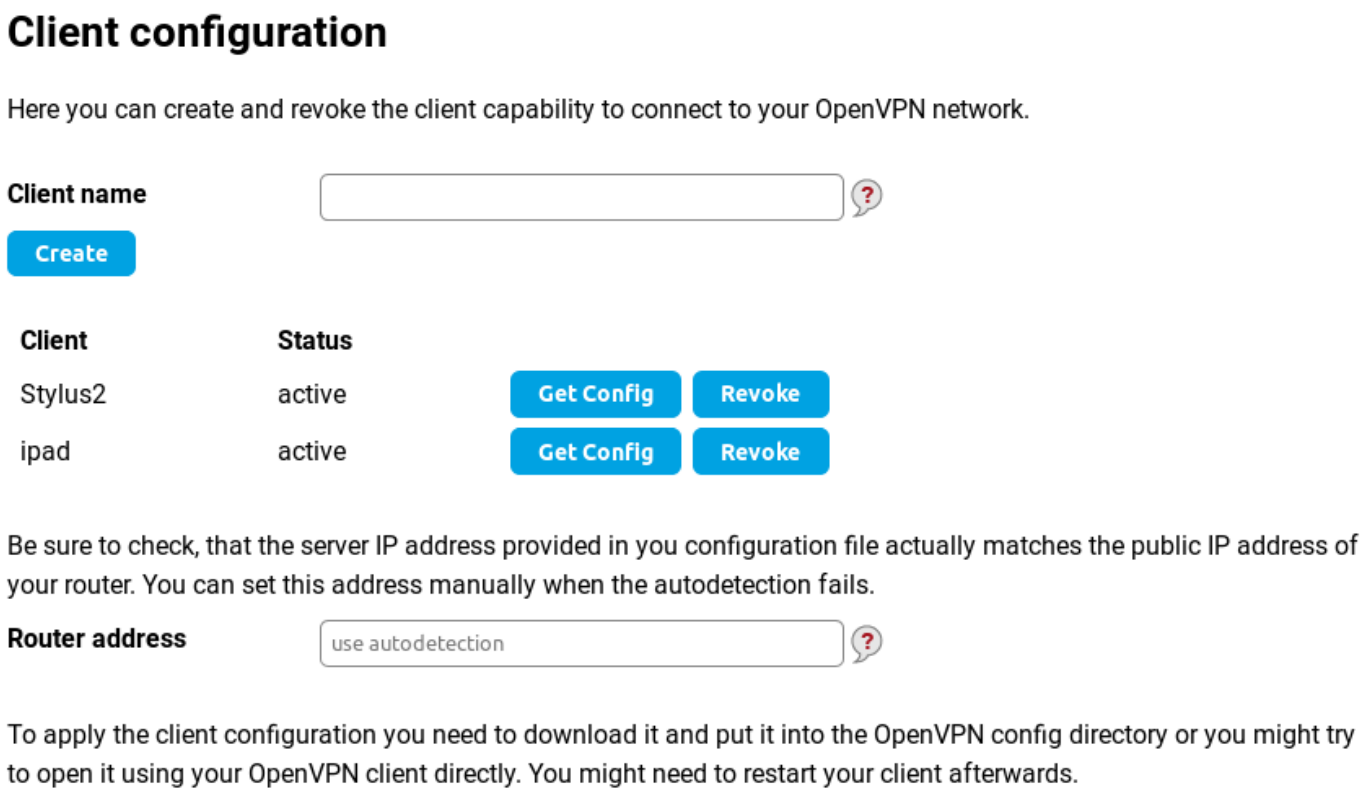
```
option ca '/etc/ssl/ca/openvpn/ca.crt'
option crl_verify '/etc/ssl/ca/openvpn/ca.crl'
option cert '/etc/ssl/ca/openvpn/01.crt'
option key '/etc/ssl/ca/openvpn/01.key'
option dh '/etc/dhparam/dh-default.pem'
```

OpenVPN arbeitet nur auf Zertifikatsbasis. Unter Turris/Foris werden die Clientzertifikate auf dem Server erzeugt und dann - ggf. mit Nachbearbeitung - auf die Clients gebracht. Dateien pro Client:

```
-rw-r--r--    1 root     root          6911 Sep  3 01:04 02.crt
-rw-r--r--    1 root     root          1582 Sep  3 01:04 02.csr
-r--------    1 root     root          3272 Sep  3 01:04 02.key
-rw-r--r--    1 root     root          6911 Sep  3 01:04 02.pem
```

OpenVPN (4/4)

Die mit „Get Config“ herunterladbare Datei enthält sowohl Konfigurationsparameter als auch den öffentlichen sowie den privaten Schlüssel. Das ist eher unüblich (unsicher?), erspart aber dem Client, einen privaten Schlüssel selbst erzeugen zu müssen.



DNSSEC (1/2)

Damit DNS-Abfragen als authentisch überprüft werden können, gibt es den Standard DNSSEC (Domain Name System Security Extensions), siehe RFC 3833.

Ein DNS-Teilnehmer kann verifizieren, dass die erhaltenen DNS-Zonendaten tatsächlich identisch sind mit denen, die der Ersteller der Zone autorisiert hat. DNSSEC wurde als Mittel gegen Cache Poisoning (Fälschen von DNS-Antworten) entwickelt. Es sichert die Übertragung von Resource Records durch digitale Signaturen ab. Eine Authentifizierung von Servern oder Clients findet nicht statt.

Vertraulichkeit ist bei DNSSEC nicht vorgesehen, DNS-Daten werden daher nicht verschlüsselt.

Seit 2010 ist DNSSEC auf allen 13 Rootservern eingeführt.

Die Verifikation mus nicht zwingend in jedem Endgerät passieren, sondern beim DNS-Resolver (i.d.R. beim Provider).

Es gibt für jede Zone einen Hauptschlüssel (KSK, Key Signing Key) und einen Arbeitsschlüssel (ZSK, Zone Signing Key). Ganz oben steht der Root-KSK. Näheres bei Heise.

DNSSEC (2/2)

Damit Nameserver möglichst wenig belastet werden, hat man sich für kurze krypto- grafische Schlüssel entschie- den – mit der Folge, dass diese immer mal wieder getauscht werden müssen, damit Angreifer möglichst wenig Zeit zum Knacken bekommen.

Aktuell soll erstmals seit 2010 der Root-KSK getauscht werden. Dieser Tausch wurde jetzt von ICANN verschoben (vom 11.10.17 mindestens nach Q1 2018), weil Tests ergeben haben, dass viele Provider den neuen Schlüssel noch nicht eingespielt haben, und auch das Protokoll, was einen automatischen Wechsel ermöglichen würde, nicht implementiert haben. Eine bestimmte Resolver-Software (Name nicht bekannt) kann das überhaupt noch nicht.

Qualifizierte elektronische Signatur

Beschreibung auf Wikipedia.

Rechtlich gesehen wird eine QeS benötigt, wenn gesetztlich für einen Vorgang die Schriftform (nicht zu verwechseln mit der Textform) vorgeschrieben ist. Beispiel: Grundstücksgeschäfte.

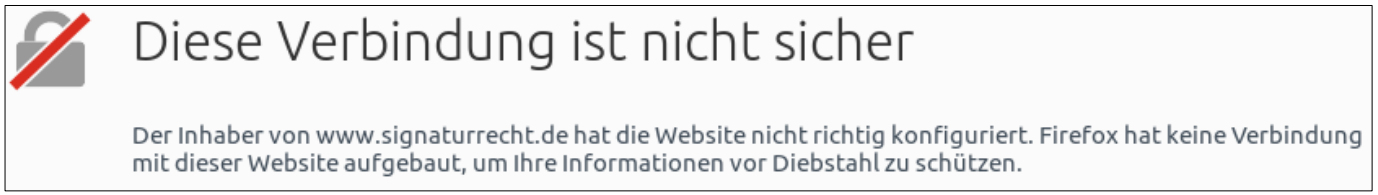
Tief blicken lässt die offizielle Liste der Anbieter bei der Bundesnetzagentur, insbesondere die Liste der ehemaligen Anbieter.

Wer die eID-Funktion des Personalausweises nutzt, braucht ein Lesegerät in der teuren Komfort-Ausführung. Das Zertifikat kostet jedes Jahr neu.

Hier ein Erfahrungsbericht von 2014. Spricht Bände. Für Privatleute völlig uninteressant.

Nützliche Links

- Grundsatzartikel Wikipedia zu Zertifikaten
  - `https://de.wikipedia.org/wiki/Digitales_Zertifikat`
- Grundsatzartikel Wikipedia zu Signaturen
  - `https://de.wikipedia.org/wiki/Digitale_Signatur`
-
- Signaturrecht



  - `https://www.signaturrecht.de/`
- Pros und Cons
  - Blog einer Firma, die mit Zertifikaten Geld verdient (Beispielbeitrag)
  - Dan Kaminski, Security-Experte (Vortrag CCC 2009)