TLS / SSL

- OpenSSL - TLS-/SSL-Zertifikate
- Diffie-Hellman Parameterdatei Bitgröße ermitteln
- (Zwischen-)Zertifikat Key und Zertifikatsrequest per Script auf Plausibilität prüfen
- CAcert Root Zertifikat unter Debian/Ubuntu einbinden
- Signatur-Algorithmen einer Zertifikatskette anzeigen

OpenSSL - TLS-/SSL-Zertifikate

OpenSSL - Zertifikate anzeigen/prüfen/testen

- Zertifikat komplett anzeigen

```
openssl x509 -noout -text -in <zertifikatsname.crt>
```

- den Herausgeber des Zertifikats anzeigen

```
openssl x509 -noout -issuer -in <zertifikatsname.crt>
```

- Für wen wurde das Zertifikat ausgestellt?

```
openssl x509 -noout -subject -in <zertifikatsname.crt>
```

- Für welchen Zeitraum ist das Zertifikat gültig?

```
openssl x509 -noout -dates -in <zertifikatsname.crt>
```

- das obige kombiniert anzeigen

```
openssl x509 -noout -issuer -subject -dates -in <zertifikatsname.crt>
```

- den hash anzeigen

```
openssl x509 -noout -hash -in <zertifikatsname.crt>
```

- den MD5-Fingerprint anzeigen

```
openssl x509 -noout -fingerprint -in <zertifikatsname.crt>
```

- ein SSL-Zertifikat prüfen

```
openssl verify -verbose <zertifikatsname.crt>
```

- einen SSL-Port auf Zertifikate abfragen (Beispiel LDAP)

```
echo QUIT | openssl s_client -connect localhost:636 -showcerts
```

- ein HTTPS-Serverzertifikat runterladen

```
echo QUIT | openssl s_client -connect www.magenbrot.net:443 | sed -ne '/BEGIN CERT/,/END CERT/p'
```

- ein HTTPS-Serverzertifikat runterladen und in lesbar ausgeben

```
echo QUIT | openssl s_client -connect www.magenbrot.net:443 | openssl x509 -noout -text
```

- Gültigkeit eines HTTPS-Serverzertifikats anzeigen

```
echo QUIT | openssl s_client -connect www.magenbrot.net:443 2>/dev/null | openssl x509 -noout -d
```

- bei Webservern mit SNI muss der gewünschte Host im Header mitgeschickt werden, da sonst das SSL-Zertifikat des Default-Vhosts ausgeliefert wird

```
echo QUIT | openssl s_client -tls1_2 -servername ovtec.it -connect ovtec.it:443 | openssl x509 -
```

- das Gleiche für Mailservices mit STARTTLS (SMTP, IMAP, POP3)

```
# herunterladen und in Datei speichern:
echo QUIT | openssl s_client -starttls smtp -crlf -connect mailgw.ovtec.it:25 | sed -ne '/BEGIN
echo QUIT | openssl s_client -starttls pop3 -crlf -connect mail.ovtec.de:110 | sed -ne '/BEGIN C
echo QUIT | openssl s_client -starttls imap -crlf -connect mail.ovtec.de:143 | sed -ne '/BEGIN C

# in lesbar ausgeben:
echo QUIT | openssl s_client -starttls smtp -crlf -connect mailgw.ovtec.it:25 | openssl x509 -no
echo QUIT | openssl s_client -starttls pop3 -crlf -connect mail.ovtec.de:110 | openssl x509 -noo
echo QUIT | openssl s_client -starttls imap -crlf -connect mail.ovtec.de:143 | openssl x509 -noo
```

CSR erzeugen/anzeigen (Certificate Signing Request)

Wenn Ihr ein offizielles Zertifikat bestellen wollt, müsst ihr für die Certificate Authority (CA) einen Zertifikatsantrag erstellen. Dieser wird auf der Webseite Eurer CA (z.B. Thawte, Geotrust, Startssl) hochgeladen und die CA erzeugt daraus dann das endgültige Zertifikat, das ihr dann zusammen mit den Zwischenzertifikaten (intermediate-certificate) herunterladen könnt.

Die aktuellen Browser verlangen die Angabe der DNS-Namen, die im Zertifikat enthalten sein sollen, in einer X509v3 Erweiterung zu OpenSSL als sogenannte „Subject Alternative Names“ (SAN). Der CommonName alleine reicht nicht mehr aus.

Umlaute im Städte- oder Firmennamen sind mit utf8 auch kein Problem mehr.

Ich habe eine Konfigurationsdatei erstellt, die ihr herunterladen und für eure Zwecke anpassen könnt. Das erstellen eines 4096-Bit Keys und passender CSR ist dann kein Problem mehr und mit einem Kommando erledigt:

```
# Konfigurationsdatei herunterladen und anpassen
wget -O request.cnf https://wiki.magenbrot.net/_media/linux/kryptographie/ssl/request.cnf
vi request.cnf # SAN anpassen!

# 4096-bit Key erstellen und das CSR mit den Angaben aus der Konfigurationsdatei generieren
openssl req -nodes -newkey rsa:4096 -keyout ovtec.it.key -new -out ovtec.it.csr -config request.

# CSR anzeigen lassen (mit UTF8)
openssl req -noout -text -nameopt utf8 -in ovtec.it.csr
```

- ALTE Anleitung: mit diesen Kommandos könnt ihr einen Key und das dazu gehörende CSR erstellen

```
# 4096 Bit RSA-Key erzeugen
openssl genrsa -out <zertifikatsname.key> 4096

# den CSR dazu erzeugen
openssl req -new -sha256 -key <zertifikatsname.key> -out <zertifikatsname.csr>

#jetzt sind ein paar Fragen zu beantworten (gibt man nur einen . ein so bleibt das Feld leer):
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:Bayern
Locality Name (eg, city) []:Fuerth
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Deine Firma
Organizational Unit Name (eg, section) []:.
```

```
Common Name (eg, YOUR name) []:www.meinedomain.de
Email Address []:webmaster@meinedomain.de

Please enter the following 'extra' attributes to be sent with your certificate request
A challenge password []:
An optional company name []:
```

(optional) den Key mit einer Passphrase versehen. Diese Passphrase wird dann z.B. beim Starten von Apache abgefragt. Also Achtung: Ein automatischer Start des Apachen ist dann nur noch mit weiteren Tricks möglich.

```
openssl rsa -des3 -in <zertifikatsname.key> -out <zertifikatsname.key.sec>
```

- einen CSR (Zertifikatsrequest) anzeigen

```
openssl req -noout -text -in <request.csr>
```

Passphrase entfernen/ändern

- Passphrase für ein Keyfile entfernen/ändern (RSA Keys)

```
# Passphrase entfernen
openssl rsa -in <zertifikatsname.key> -out <neueskeyfile.key>


# Passphrase ändern
openssl rsa -des3 -in <zertifikatsname.key> -out <neueskeyfile.key>
```

- Passphrase für ein Keyfile entfernen/ändern (ECC Keys)

```
# Passphrase entfernen
openssl pkey -in <zertifikatsname.key> -out <neueskeyfile.key>


# Passphrase ändern
openssl pkey -des3 -in <zertifikatsname.key> -out <neueskeyfile.key>
```

Selbstsignierte (selfsigned) Zertifikate erstellen

Diese Zertifikate können für interne Zwecke eingesetzt werden oder für den Zeitraum bis man von der Trusted CA sein richtiges Zertifikat bekommt. Mit wenigen Schritten ist ein solches Zertifikat erstellt, diese Beispiel erzeugt ein für 60 Tage gültiges Zertifikat:

```
openssl genrsa -out <zertifikatsname.key> 4096
[...]
openssl req -new -sha256 -key <zertifikatsname.key> -out <zertifikatsname.csr> #(siehe oben)
[...]
openssl x509 -req -sha256 -days 60 -in <zertifikatsname.csr> -signkey <zertifikatsname.key> -out
[...]
```

oder als komfortabler Einzeiler:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -sha256 -nodes -subj
```

Zertifikate konvertieren

- PEM nach DER

```
openssl x509 -outform der -in certificate.pem -out certificate.der
```

- PEM nach P7B

```
openssl crl2pkcs7 -nocrl -certfile certificate.cer -out certificate.p7b -certfile CACert.cer
```

- PEM nach PKCS12 (P12)

```
openssl pkcs12 -export -out certificate.p12 -inkey userkey.pem -in usercert.pem
```

- PEM nach PFX

```
openssl pkcs12 -export -out certificate.pfx -inkey privateKey.key -in certificate.crt -certfile
```

- DER nach PEM

```
openssl x509 -inform der -in certificate.cer -out certificate.pem
```

- P7B / PKCS7 nach PEM

```
openssl pkcs7 -print_certs -in certificate.p7b -out certificate.cer
```

- P7B / PKCS7 nach PFX

```
openssl pkcs7 -print_certs -in certificate.p7b -out certificate.cer
openssl pkcs12 -export -in certificate.cer -inkey privateKey.key -out certificate.pfx -certfile
```

- PFX / PKCS12 nach PEM

```
# es wird nach dem Import-Passwort gefragt, wenn eines gesetzt wurde:
openssl pkcs12 -in certificate.pfx -out certificate.pem -nodes

# beim Fehler "Mac verify error: invalid password?" bitte mit der nächsten Methode probieren
```

- PFX / PKCS12 nach PEM (wenn er das Passwort nicht frisst):

```
openssl pkcs12 -in certificate.pfx -out certificate.pem  -nodes -password pass:<PASSWORT>
```

PEM-Datei richtig erstellen

Seit Apache 2.4.8 ist SSLCertificateChainFile als deprecated markiert. Das heißt, dass nun Zertifikat und Zwischenzertifikat in einer gemeinsamen Datei gespeichert werden müssen und über die Direktive SSLCertificateFile geladen werden.

Dabei ist die Reihenfolge wichtig. Eine solche Datei wird folgendermaßen erstellt:

```
cat meine-domain.de.crt ssl-ca-intermediate.crt > meine-domain.de-bundle.crt
```

Ganz selten wird auch das Root der SSL-CA benötigt, das hängt dann einfach ganz am Ende dran:

```
cat meine-domain.de.crt ssl-ca-intermediate.crt ssl-ca-root.crt > meine-domain.de-bundle.cr
```

Um solche Dateien einfach zu erkennen, füge ich ein -bundle in den Dateinamen ein.

PEM inklusive private Key

Es kann auch noch der SSL-Key eingefügt werden. Dieser kommt dann an erster oder letzter Stelle. Die Zertifikate übrigen in der üblichen Reihenfolge (Zertifikat, Intermediate und gegebenenfalls noch das Root-Zertifikat). Apache unterstützt das zwar, es wird allerdings aktuell nicht empfohlen.

```
cat meine-domain.de.key meine-domain.de.crt ssl-ca-intermediate.crt (ssl-ca-root.crt) > me
```

Folgender Einzeiler baut das entsprechende PEM-File zusammen, falls Key und Zertifikat in zwei Dateien mit den Endungen .key und .crt vorliegen:

```
for i in `ls *.key`; do NAME=$(echo $i | sed 's/.key//'); echo $NAME; cat $NAME.crt $NAME.key >>
```

Prüfen ob ein Zertifikat zu einem Key passt

Der private Teil eines Schlüssels enthält verschiedene Zahlen. Zwei dieser Zahlen bilden den „Publiy Key“ (diese Zahlen sind dann auch im Zertifikat erhalten), der Rest gehört zum „Private Key“. Um zu prüfen, ob der public key zum private key passt, können diese beiden Zahlen ausgelesen und verglichen werden.

```
# Zertifikate
$ openssl x509 -noout -modulus -in server.crt | openssl md5
# Private Schlüssel
$ openssl rsa -noout -modulus -in server.key | openssl md5
# Zertifikatsanforderung
$ openssl req -noout -modulus -in server.csr | openssl md5
```

oder als einfach zu verwendendes Script:

```
#!/bin/bash
#
# check if certificate, signing request and key match
#
# $Id: check-certificate.sh 524 2016-01-15 11:30:11Z magenbrot $
# zlFamous added intermediate and key size check
#

if [ "x$1" = "x" ]; then
  echo "Usage: $0 <filename without .key, .crt, .csr or .intermediates>"
  exit 1
fi

if [ -e $1.key ]; then
  output="$1.key: `openssl rsa -noout -modulus -in $1.key | openssl md5 | cut -d" " -f2`"
  key_size=`openssl rsa -noout -text -in $1.key | grep "Private-Key" | cut -d" " -f2 | cut -d"("
  if [ $key_size -lt 4096 ]; then
    output="$output \e[39m(key size: \e[33m$key_size\e[39m bit)"
  else
    output="$output \e[39m(key size: \e[32m$key_size\e[39m bit)"
  fi
  echo -e $output
```

```
else
  echo "$1.key: file not found"
fi

if [ -e $1.csr ]; then
  echo -n "$1.csr: "
  openssl req -noout -modulus -in $1.csr | openssl md5 | cut -d" " -f2
else
  echo "$1.csr: file not found"
fi

if [ -e $1.crt ]; then
  echo -n "$1.crt: "
  openssl x509 -noout -modulus -in $1.crt | openssl md5 | cut -d" " -f2
else
  echo "$1.crt: file not found"
fi

if [ -e $1.intermediates ]; then
  echo -n "$1.intermediates: "
  subject=`openssl x509 -noout -subject_hash -in $1.intermediates`
  issuer=`openssl x509 -noout -issuer_hash -in $1.crt`
  if [ "$subject" != "" -o "$issuer" != "" ]; then
    if [ "$subject" == "$issuer" ]; then
      signature=`openssl x509 -noout -text -in $1.intermediates | grep "Signature Algorithm:" |
      echo -e "\e[32missuer matches subject \e[39m- signature hash: \e[32m$signature\e[39m"
    else
      echo -e "\e[31missuer doesn't match subject"
    fi
  fi
else
  echo "$1.intermediates: file not found"
fi
```

Diffie-Hellman Parameterdatei Bitgröße ermitteln

Auf Diffie-Hellman aufsetzende Cipher benötigen eine entsprechende Parameterdatei. Die Standardgröße beträgt 1024 bit. Die Empfehlung nach der Logjam-Attacke sind mind. 2048 bit, besser 4096 bit.

Die Parameterdatei wird folgendermaßen erstellt:

```
openssl dhparam -out dhparams.pem 4096
```

Aus der Datei läßt sich die Bitgröße nicht auf Anhieb herauslesen. Mit diesem Befehl lässt sich die Info darstellen:

```
openssl dhparam -inform PEM -in dhparams.pem -check -text
```

(Zwischen-)Zertifikat Key und Zertifikatsrequest per Script auf Plausibilität prüfen

Dieses Script prüft folgende Punkte:

- mind. 4096 Bit Key
- Passen CSR, CRT, KEY zusammen (openssl modulus / md5)?
- Signatur-Hash prüfen
- Passt das Intermediate-CRT zum CRT

Namenskonvention der Dateien:

- meine-domain.de.key
- meine-domain.de.crt
- meine-domain.de.csr
- meine-domain.de.intermediates

```
#!/bin/bash
#
# check if certificate, signing request and key match
#

if [ "x$1" = "x" ]; then
  echo "Usage: $0 <filename without .key, .crt, .csr or .intermediates>"
  exit 1
fi

if [ -e $1.key ]; then
  output="$1.key: `openssl rsa -noout -modulus -in $1.key | openssl md5 | cut -d" " -f2`"
  key_size=`openssl rsa -noout -text -in $1.key | grep "Private-Key" | cut -d" " -f2 | cut -
d"(" -f2`
  if [ $key_size -lt 4096 ]; then
    output="$output \e[39m(key size: \e[33m$key_size\e[39m bit)"
  else
    output="$output \e[39m(key size: \e[32m$key_size\e[39m bit)"
  fi
  echo -e $output
else
  echo "$1.key: file not found"
fi
```

```
if [ -e $1.csr ]; then
  echo -n "$1.csr: "
  openssl req -noout -modulus -in $1.csr | openssl md5 | cut -d" " -f2
else
  echo "$1.csr: file not found"
fi

if [ -e $1.crt ]; then
  echo -n "$1.crt: "
  openssl x509 -noout -modulus -in $1.crt | openssl md5 | cut -d" " -f2
else
  echo "$1.crt: file not found"
fi

if [ -e $1.intermediates ]; then
  echo -n "$1.intermediates: "
  subject=`openssl x509 -noout -subject_hash -in $1.intermediates`
  issuer=`openssl x509 -noout -issuer_hash -in $1.crt`
  if [ "$subject" != "" -o "$issuer" != "" ]; then
    if [ "$subject" == "$issuer" ]; then
      signature=`openssl x509 -noout -text -in $1.intermediates | grep "Signature Algorithm:"
| cut -d" " -f7 | head -n1`
      echo -e "\e[32missuer matches subject \e[39m- signature hash: \e[32m$signature\e[39m"
    else
      echo -e "\e[31missuer doesn't match subject"
    fi
  fi
  chown root:root $1.intermediates
  chmod 0600 $1.key $1.csr $1.crt $1.intermediates
else
  echo "$1.intermediates: file not found"
fi
```

CAcert Root Zertifikat unter Debian/Ubuntu einbinden

Da das CAcert Root Zertifikat leider nicht mehr mit Debian und damit auch nicht mit Ubuntu ausgeliefert wird folgt hier eine Anleitung, wie man es im System einbindet. Das betrifft nur das Betriebssystem, ggf. muss das Zertifikat auch in euren Browser importiert werden.

Dazu müssen die CAcert Root Zertifikate (root.crt und class3.crt) in einem Verzeichnis unterhalb von /usr/local/share/ca-certificates abgelegt werden und das Zertifikatsbundle neu erzeugt werden.

```
mkdir /usr/local/share/ca-certificates/cacert.org
wget -P /usr/local/share/ca-certificates/cacert.org http://www.cacert.org/certs/root.crt http://
update-ca-certificates
```

Signatur-Algorithmen einer Zertifikatskette anzeigen

Da mit SHA1 signierte Zertifikate inzwischen als unsicher eingestuft werden, habe ich nach einer einfachen Möglichkeit gesucht, wie ich herausfinden kann, welche meiner Zertifikate davon betroffen sind.

Es reicht allerdings nicht, nur das Serverzertifikat auszutauschen. Es sollte auch die Zertifikatskette (Certificate chain) untersucht werden, da ggf. auch das Zwischen- und CA-Zertfikat ausgetauscht werden muss. Wobei das CA-Zertifikat natürlich nur von der CA, bzw. dem Herausgeber selbst (Geotrust, Thawte, etc) ausgetauscht werden kann.

Aufgerufen wird das Script so (kann dann z.B. in einer Schleife mit euren verschiedenen Servern/Ports gefüttert werden):

```
# ./check-ssl-chain.sh www.heise.de:443
    Signature Algorithm: sha256WithRSAEncryption
        Subject: C=DE, ST=Niedersachsen, L=Hannover, O=Heise Zeitschriften Verlag GmbH und Co KG
    Signature Algorithm: sha256WithRSAEncryption

    Signature Algorithm: sha256WithRSAEncryption
        Subject: C=US, O=thawte, Inc., CN=thawte SHA256 SSL CA
    Signature Algorithm: sha256WithRSAEncryption
```

oder:

```
# ./check-ssl-chain.sh google.com:443
    Signature Algorithm: sha1WithRSAEncryption
        Subject: C=US, ST=California, L=Mountain View, O=Google Inc, CN=google.com
    Signature Algorithm: sha1WithRSAEncryption

    Signature Algorithm: sha1WithRSAEncryption
        Subject: C=US, O=Google Inc, CN=Google Internet Authority G2
    Signature Algorithm: sha1WithRSAEncryption

    Signature Algorithm: sha1WithRSAEncryption
        Subject: C=US, O=GeoTrust Inc., CN=GeoTrust Global CA
    Signature Algorithm: sha1WithRSAEncryption
```

Das Beispiel 1 (Heise) zeigt, dass der Admin fleissig war und die Zertifikate schon gegen SHA256-signierte ausgetauscht hat. Google hingegen setzt noch mit SHA1 signierte Zertifikate ein.

Hier noch das Script:

```
#!/bin/sh

HOST=$1
TMP=`mktemp -d`

echo QUIT | openssl s_client -showcerts -connect $HOST 2>/dev/null | sed -ne
'/BEGIN CERT/,/END CERT/p' | awk -v TMP="$TMP" '/BEGIN/{n++}{print >TMP"/out" n ".crt" }'

for i in `ls $TMP/out*`; do
  openssl x509 -in $i -noout -text | egrep "Signature Algorithm|Subject:"
  echo
done

rm -rf $TMP
```