# Das OpenSSL Handbuch

DFN-PCA, Vogt-Kölln-Strasse 30, Universität Hamburg, FB Informatik - RZ, D-22527 Hamburg

Version 2.03 18.04.2000

OpenSSL ist eine frei verfügbare Implementierung des SSL/TLS-Protokolls und bietet zahlreiche Funktionen zur X.509-Zertifikat-Verwaltung sowie verschiedene kryptographische Funktionen. Es basiert auf dem SSLeay-Paket, das von Eric A. Young und Tim Hudson entwickelt wurde. OpenSSL wird heute von einer unabhängigen Gruppe weiterentwickelt. Die folgenden Seiten geben detaillierte Informationen zur Installation und zum Einsatz von `openssl-0.9.5-dev`. Hinweise und Links zur *Verfügbarkeit der Software* <`http://www.pca.dfn.de/dfnpca/certify/ssl/handbuch/links.html`> finden Sie auf einer eigenen Seite.

## Inhaltsverzeichnis

# 1 HINWEIS

Dieses Dokument (das „OpenSSL Handbuch“) entstand in einem DFN-Projekt an der Universität Hamburg und wurde nach bestem Wissen und Gewissen verfaßt. Dennoch wird keine Haftung für die Korrektheit, Vollständigkeit oder Anwendbarkeit der hier beschriebenen Informationen und der vorgeschlagenen Maßnahmen übernommen. Ferner kann keine Haftung für eventuelle Schäden, entstanden durch die Anwendung der in diesem Dokument beschriebenen Anweisungen, übernommen werden. Die Verantwortung für die Verwendung der hier beschriebenen Verfahren und Programme liegt allein bei den die Installation durchführenden Personen.

# 2 Einleitung

OpenSSL ist eine frei verfügbare Implementierung des *SSL/TLS-Protokolls* <`http://www.pca.dfn.de/eng/team/ske/pem-dok.html#SSL`> und bietet zusätzlich Funktionen zur Zertifikat-Verwaltung sowie verschiedene kryptographische Funktionen. Es basiert auf dem SSLeay-Paket, das von Eric A. Young und Tim Hudson entwickelt wurde. OpenSSL als Nachfolger von SSLeay wird derzeit von einer unabhängigen Gruppe weiterentwickelt.

Das Paket umfaßt mehrere Applikationen, z.B. zur Erzeugung von Zertifikaten, von Zertifizierungsanträgen und zur Verschlüsselung. Diese einzelnen Applikationen sind zusammengefaßt in einem Kommandozeilen-Programm: `openssl`.

Der Schwerpunkt dieses Dokuments liegt auf dem praktischen Einsatz von OpenSSL zur Erzeugung, Verwaltung und Verwendung von Zertifikaten. Die Funktionen zur Zertifikatverwaltung in OpenSSL sind ursprünglich nicht für den Betrieb einer CA gedacht gewesen. Es sollten lediglich Zertifikate erzeugt werden können, um das (zertifikatbasierte) SSL/TLS-Protokoll sinnvoll einsetzen zu können. Die Entwicklung des OpenSSL-Teils, der für die Zertifikatverwaltung notwendig ist, ist inzwischen soweit vorangeschritten, daß der Betrieb kleiner bis mittlerer CAs gut möglich ist.

Die letzte OpenSSL *Anwender*-Version (0.9.4) stammt vom 9.6.99. Dieses Dokument bezieht sich auf die OpenSSL-*Entwickler*-Version 0.9.5-*dev*. Die in diesem Dokument gemachten Angaben sollten sich problemlos auf die nächste Anwender-Version beziehen lassen. Nach Herausgabe der neuen Anwender-Version wird dieses Dokument ggf. angepaßt werden. Die Entwickler-Versionen, „snapshot“ genannt, tragen die Bezeichnung `openssl-SNAP-2000`*mmdd*, wobei *mm* und *dd* für die Monats- bzw. Tages-Zahl stehen. Die `SNAP`-Bezeichnung wird in diesem Dokument immer dann verwendet, wenn es um konkrete Kommandos zu Übersetzung bzw. Installation geht. Andernfalls wird die Bezeichnung `openssl-0.9.5-dev` verwendet.

Die Entwickler-Version zeichnet sich gegenüber der letzten Anwender-Version u.a. durch eine deutlich verbesserte Dokumentation aus. Für viele Teile des Programms sind jetzt Manual-Seiten vorhanden.

# 3 Installation von OpenSSL-0.9.5-dev

Es gibt zwei Möglichkeiten, das Paket zu kompilieren. Die erste Möglichkeit faßt die einzelnen Applikationen zu einem monolithischen Programm zusammen, `openssl`. Auf die einzelnen Applikationen wird dann zugegriffen, indem `openssl` mit dem Applikationsnamen als Parameter aufgerufen wird.

Die zweite Möglichkeit kompiliert die Applikationen als eigenständige Programme, es gibt kein monolithisches Programm.

Für beide Möglichkeiten ist die Konfiguration und teilweise auch die Übersetzung und Installation des Paketes gleich. Daher wird ggf. beides gemeinsam behandelt.

Alle Angaben zur Übersetzung und Konfiguration des Paketes beziehen sich auf das Betriebssystem SunOS 5.5.1 (Solaris 2.5.1), den C-Compiler gcc-2.7.2.1 und `openssl-SNAP-20000222.tar.gz`.

## 3.1 Konfiguration und Übersetzung

Zunächst wird das OpenSSL-Quell-Paket ausgepackt, z.B. in `/usr/src`:

```
gzip -dc openssl-SNAP-2000mmdd.tar.gz | tar -xvf -
```

Nach Wechsel in das Quell-Verzeichnis `openssl-SNAP-2000mmdd` wird das Paket konfiguriert. Zur Konfiguration können mehrere Optionen angegeben werden. Hier wird nur auf zwei Optionen eingegangen, die anderen

werden in der Datei `openssl-SNAP-2000mmdd/INSTALL` beschrieben. Mit der Option `--prefix=`*xxx* wird ein Pfad angegeben, unterhalb dessen die OpenSSL Programm-Dateien installiert werden. Es werden dann bei einer automatischen Installation die Verzeichnisse *xxx*`/bin`, *xxx*`/lib` und *xxx*`/include/openssl` angelegt. Mit der Option `--openssldir=`*xxx* wird festgelegt, in welchem Verzeichnis die Dateien bzw. Verzeichnisse zur Zertifikatverwaltung und die Manual-Seiten abgelegt werden. Es werden dann die Verzeichnisse *xxx*`/certs`, *xxx*`/misc`, *xxx*`/private`, *xxx*`/man` und die Konfigurationsdatei *xxx*`/openssl.cnf` angelegt.

Konfiguriert wird das Paket durch folgenden Befehl:

```
sh ./config --prefix=/usr/local --openssldir=/usr/local/etc/ssl
```

Nun kann das Paket übersetzt werden. Durch die Übersetzung werden Bibliotheken erzeugt, die auch für die Erzeugung der Programmsammlung (siehe unten (3.2)) benötigt werden. Daher sind die folgenden Befehl auch auszuführen, wenn nur die Programmsammlung erzeugt werden soll.

```
make
```

Mit folgendem Befehl werden umfangreiche Tests des übersetzten Pakets durchgeführt:

```
make test
```

Erfolgte der Testlauf im Sinne der Tests fehlerfrei, kommt abschließend eine Meldung ähnlich der folgenden:

```
OpenSSL 0.9.5-dev 09 Aug 1999
built on: Mon Feb 14 10:44:30 MET 2000
platform: solaris-sparcv8-gcc
options:  bn(64,32) md2(int) rc4(ptr,char) des(idx,cisc,16,long) idea(int) blowfish(ptr)
compiler: gcc -DTHREADS -D_REENTRANT -mv8 -O3 -fomit-frame-pointer -Wall -DB_ENDIAN -DBN_DIV2W
'test' is up to date.
```

Soll das monolithische Programm eingesetzt werden, kann jetzt mit der Installation, die im übernächsten Abschnitt (3.3) beschrieben wird, fortgefahren werden.

Zur Erzeugung der Programmsammlung ist eine weiterer Übersetzungsschritt notwendig. Er wird im nächsten Abschnitt (3.2) beschrieben.

## 3.2 Übersetzung als Programmsammlung

Nach Ausführung des `make`-Befehls im vorigen Abschnitt (3.1) liegt jetzt das komplette Paket übersetzt vor. Die Übersetzung diente aber nur dazu, die Bibliotheken `libssl.a` und `libcrypto.a` im Quellverzeichnis zu erzeugen. Sie sind notwendig für die weitere Übersetzung. Die einzelnen Applikationen werden am einfachsten mit folgendem Shell-Skript im Quell-Verzeichnis übersetzt. Da die einzelnen Applikation z.T. unterschiedliche Programmteile und Bibliotheken benötigen, fällt die Übersetzung für jedes Programm etwas unterschiedlich aus. Die benötigten Bibliotheken werden statisch in die Programme gelinkt, so daß sie Programme jeweils um die fünf Megabyte groß sind.

Vor der Ausführung des Skriptes muß ein Verzeichnis `out` angelegt werden. Im Skript wird angenommen, daß `out` und das OpenSSL-Quell-Verzeichnis im selben Verzeichnis liegen, z.B. `/usr/src`. Außerdem muß in der Datei `openss-SNAP-2000mmdd/apps/app_rand.c` eine kleine Änderung vorgenommen werden:

Die folgende Zeile in `app_rand.c`

```
112  #include "apps.h"
```

ersetzen durch

```
112  #define NON_MAIN
113  #include "apps.h"
114  #undef NON_MAIN
```

Nach dem Wechsel in das Verzeichnis `/usr/src` und Anlegen des Verzeichnisses `/usr/src/out`, kann das folgende Skript ausgeführt werden. Das Skript funktioniert mit der OpenSSL-Version `openssl-0.9.5beta2`

```
#!/bin/sh

FLAGS='-mv8 -O3 -fomit-frame-pointer -Icrypto -Iinclude -Wall'

cd openssl-0.9.5beta2

for i in speed verify version ; do
    gcc ${FLAGS} -o ../out/$i apps/$i.c libcrypto.a
done

for i in asn1pars crl crl2p7 dgst dh dsa enc nseq pkcs7 rsa ; do
    gcc ${FLAGS} -o ../out/$i apps/$i.c apps/apps.c libcrypto.a
done

mv ../out/asn1pars ../out/asn1parse
mv ../out/crl2p7   ../out/crl2pkcs7

gcc ${FLAGS} -o ../out/pkcs8 apps/pkcs8.c apps/apps.c apps/app_rand.c \
    libcrypto.a -lsocket

gcc ${FLAGS} -o ../out/errstr apps/errstr.c apps/apps.c \
     libssl.a libcrypto.a

for i in ciphers sess_id ; do
    gcc ${FLAGS} -o ../out/$i apps/$i.c apps/apps.c \
           libssl.a libcrypto.a -lsocket
done

for i in ca dhparam dsaparam gendh gendsa genrsa passwd pkcs12 \
         req smime x509 ; do
    gcc ${FLAGS} -o ../out/$i apps/$i.c apps/apps.c apps/app_rand.c \
        libssl.a libcrypto.a -lsocket
done

gcc ${FLAGS} -o ../out/s_time apps/s_time.c apps/app_rand.c apps/s_cb.c \
    libssl.a libcrypto.a -lsocket -lnsl
```

```
for i in s_client s_server ; do
    gcc ${FLAGS} -o ../out/$i apps/$i.c \
        apps/app_rand.c apps/s_cb.c apps/s_socket.c \
        libssl.a libcrypto.a -lsocket -lnsl
done

cd ..
```

Für die spkac-Applikationen war die Übersetzung nicht erfolgreich.

Die Installation erfolgt durch Kopieren der Dateien. Dazu kann wie in der zweiten Hälfte des nächsten Abschnitts (3.3) beschrieben wird, vorgegangen werden. Dabei muß allerdings der dritte Punkt ersetzt werden. Statt

```
cp apps/openssl tools/c_rehash $(SSLDIR)/bin/
```

muß folgendes Kommando gegeben werden:

```
cp ../out/* tools/c_rehash $(SSLDIR)/bin/
```

## 3.3 Installation

Nachdem das Paket übersetzt wurde, kann es jetzt installiert werden. Zunächst wird eine automatischen Installation beschrieben, und dann eine alternative Installation durch Kopieren. Die Programmsammlung kann nicht automatisch installiert werden. Sie muß durch Kopieren Installiert werden.

Die automatische Installation erfolgt durch das Kommando

```
make install
```

*Achtung:*
In `$(SSLDIR)` muß noch das Verzeichnis `newcerts` angelegt werden. Der `install`-Befehl erzeugt es nicht. Das Verzeichnis wird aber von der Default-Konfigurationsdatei `openssl.cnf` (s.u.) verlangt, um dort die erzeugten Zertifikate abzulegen.

Jetzt muß noch eine Datei angelegt werden, die die aktuelle Seriennummer des herauszugebenden Zertifikats in hexadezimaler Form enthält:

```
echo "01" > $(SSLETC)/serial
```

Dann muß noch eine Indexdatei für die erzeugten Zertifikate angelegt werden:

```
touch $(SSLETC)/index.txt
```

Die beiden Dateien `serial` und `index.txt` werden nach jeder erfolgreichen Zertifizierung eines Requests durch das Programm `ca` geändert, d.h. die Seriennummer in `serial` wird um den Wert eins erhöht, und in `index.txt` wird das herausgegebene Zertifikat registriert (siehe `index.txt (7.1)`).

Es ist wichtig, daß die OpenSSL-Konfigurationsdatei `$(SSL_DIR)/lib/openssl.cnf` vor dem Benutzen der OpenSSL-Applikationen durchgesehen und den Erfordernissen angepaßt wird. (Siehe Beispiel im Anhang `opsenssl.cnf (A)`.)

Wer etwas mehr Kontrolle über die Installation haben will, kann sich an folgendes Verfahren halten (Installation durch Kopieren).

Die ausführbaren Dateien, die Header-Dateien, die Bibliotheken und die Manual-Seiten können in eine vorhandene Verzeichnisstruktur kopiert werden. Das können die entsprechenden Verzeichnisse in `/usr/local` sein (`bin`, `include`, `lib` und `man`). Für die Installation der Konfigurationsdatei und der Verzeichnisse zur Zertifikatverwaltung muß zunächst ein Verzeichnis erzeugt werden, z.B. `/usr/local/etc/ssl`. Der Pfad dieses Verzeichnisses sollte günstigerweise mit dem vor der Kompilierung angegebenen (`--openssldir`) übereinstimmen. Andernfalls funktionieren einige Applikationen nicht ganz vollständig, da der Pfad in die Bibliothek `libcrypto.a` einkompiliert wird. In diesem Verzeichnis werden dann die Verzeichnisse `crl`, `certs`, `newcerts`, `misc` und `private` erzeugt. Anschließend werden die Dateien kopiert:

*$(SSLDIR) = Installationspfad für Programmdateien*
*$(SSLETC) = Installationspfad für Dateien zur Zertifikatverwaltung*

- `cp libcrypto.a libssl.a $(SSLDIR)/lib/`
- `chmod 644 $(SSLDIR)/lib/*`
- `cp apps/openssl tools/c_rehash $(SSLDIR)/bin/`
- `chmod 755 $(SSLDIR)/bin/*`
- `cp -r include/openssl $(SSLDIR)/include/`
- `rsaref.h` wird nicht benötigt und kann gelöscht werden:
  `rm $(SSLDIR)/include/openssl/rsaref.h`
- `chmod 644 $(SSLDIR)/include/*`
- `cp tools/c_i* tools/c_hash tools/c_name apps/CA.pl apps/CA.sh apps/der_chop $(SSLETC)/misc/`
- `chmod 755 $(SSLETC)/misc/*`
- `cp apps/openssl.cnf $(SSLETC)`
- `chmod 644 $(SSLETC)/openssl.cnf`

Die Installation der Manual-Seiten ist etwas aufwendiger, da sie im `pod`-Format vorliegen. Sie müssen durch das Perl-Skript `pod2man` in Manual-Seiten gewandelt werden. Das kann durch Aufruf des entsprechenden Abschnitts im Makefile geschehen:

```
make install_docs
```

Alternativ kann auch das folgende Shell-Skript aufgerufen werden, welches eine Adaption des `install_docs`-Abschnitts ist. Die einzelnen Manual-Sektionen werden durch das Skript in `/tmp` installiert und können dann von `root` in das gewünschte Verzeichnis kopiert werden

```
#!/bin/sh

VERSION=0.9.5-dev
TOP=/usr/src/openssl-SNAP-2000mmdd

mkdir /tmp/man1 /tmp/man3 tmp/man5 /tmp/man7
```

```
for i in ${TOP}/doc/apps/*.pod; do
    cd ‘dirname $i‘
    fn=‘basename $i .pod‘
    sec=‘[ "$fn" = "config" ] && echo 5 || echo 1‘
    perl ${TOP}/util/pod2man.pl --section=$sec --center=OpenSSL \
            --release=${VERSION} ‘basename $i‘ \
            >  /tmp/man$sec/‘basename $i .pod‘.$sec
done

for i in ${TOP}/doc/crypto/*.pod ${TOP}/doc/ssl/*.pod; do
    cd ‘dirname $i‘
    fn=‘basename $i .pod‘
    sec=‘[ "$fn" = "des_modes" ] && echo 7 || echo 3‘
    perl ${TOP}/util/pod2man.pl --section=$sec --center=OpenSSL \
            --release=${VERSION} ‘basename $i‘ \
            >  /tmp/man$sec/‘basename $i .pod‘.$sec
done
```

Nun können die Seiten von `root` kopiert werden:

```
#!/bin/sh
for i in 1 3 5 7 ; do cp /tmp/man${i}/* /usr/local/man${i}/ ; end
```

Jetzt muß noch eine Datei angelegt werden, die die aktuelle Seriennummer des herauszugebenden Zertifikats in hexadezimaler Form enthält:

```
echo "01" > $(SSLETC)/serial
```

Dann muß noch eine Indexdatei für die erzeugten Zertifikate angelegt werden:

```
touch $(SSLETC)/index.txt
```

# 4 Zertifikaterweiterungen

Zertifikaterweiterungen (*Extensions*) sind von großer Bedeutung für den Umgang mit Zertifikaten. Die Extensions steuern den Verwendungszweck von Zertifikaten, sofern die Anwendungssoftware diese Extensions korrekt interpretiert. Leider ist es so, daß Anwendungssoftware und Zertifikat-Herausgeber die Verwendung und Bedeutung identischer Extensions teilweise recht unterschiedlich interpretieren. Genaueres zu dieser Problematik findet sich im lesenswerten *X.509 Style Guide* <`http://www.cs.auckland.ac.nz/~pgut001/pubs/x509guide.txt`> von Peter Gutmann.

Üblicherweise werden Extensions durch eine (P)CA beim Signieren eines Request in das dann erstellte Zertifikat gebracht. Die Bedeutung der „Certficate Standard Extensions“ wird im RFC 2459, *Internet X.509 Public Key Infrastructure - Certificate and CRL Profile* <`ftp://ftp.informatik.uni-hamburg.de/pub/doc/rfc/rfc2459.txt.gz`> beschrieben. Für den Betrieb einer CA ist es *unumgänglich*, sich mit diesen Extensions auseinanderzusetzen.

OpenSSL in der Version 0.9.5-dev unterstützt die folgenden *X.509v3-Extensions*:

- *Standard Certificate Extensions*
  - Basic Constraints (4.2)
  - Key Usage (4.3)
  - Extended Key Usage (4.4)
  - Subject Key Identifier (4.5)
  - Authority Key Identifier (4.6)
  - Subject Alternative Name (4.7)
  - Issuer Alternative Name (4.8)
  - Certificate Policies (4.9)
  - CRL Distribution Points (4.10)
- Netscape Certificate Extensions (4.11)
  *(proprietär)* `<http://home.netscape.com/eng/security/certs.html>`

Es ist auch möglich, eigene Erweiterungen zu registrieren (was Netscape mit den „Netscape Certifcate Extensions“ gemacht hat). Solange diese Erweiterungen nicht als „Critical“ markiert sind, sollte jede Anwendung, die diese Erweiterungen nicht kennt, diese ignorieren (das Zertifikat also akzeptieren).

Extensions, die von OpenSSL (noch) nicht direkt unterstützt werden, können in ihrer hexadezimalen Kodierung angegeben werden. Dazu muß die OID und die ASN.1-Syntax der Extension bekannt sein. Die ASN.1-Syntax wird dann in ihrer DER-Form (*distinguished encoding rules*, beschrieben in X.208) als hexadezimale Kodierung direkt angegeben. Auf diese Möglichkeit wird hier nicht weiter eingegangen. Genaueres dazu findet sich in der Datei `openssl.txt` im Verzeichnis `doc` der OpenSSL-Quellen.

## 4.1 Critical Bit

Das *Critical Bit* ist ein Flag, das für die meisten Extensions im Zertifikat gesetzt werden kann. Es wird beim Signieren durch eine CA zusammen mit der Extension gesetzt. Bei korrekter Implementierung einer Anwendung (z.B. eines Browsers) *muß diese eine als Critical markierte Extension interpretieren* können. Ist die Anwendung dazu nicht in der Lage (die Anwendung „kennt“ die Extension nicht), hat sie das Zertifikat, das diese Extension enthält, zurückzuweisen. Auch dann, wenn das Zertifikat technisch korrekt ist. Das Critical Bit soll also eine Möglichkeit bieten, eine bestimmte Verwendung eines Zertifikats zu erzwingen.

Da Zertifikate, die dieses Bit gesetzt haben, zurückgewiesen werden können, sollte der Einsatz des Critical Bit gut überlegt werden. Es gibt beispielsweise verschiedene Ansichten über die Verwendung der einzelnen Key Usage Attribute (s.u. (4.3)), so daß eine Critical-Markierung hier nicht ohne „Risiko“ ist.

## 4.2 Basic Constraints

Mittels Basic Constraints kann eine Anwendung erkennen, ob es sich bei einem Zertifikat um ein CA-Zertifikat handelt oder nicht. Diese Extension sollte in jedem CA-Zertifikat verwendet und als Critical markiert werden, auch wenn möglicherweise einige Anwendungen wegen der Critical-Markierung das Zertifikat zurückweisen.

Basic Constraints besteht aus einem Feld `cA`, welches ein BOOLEAN ist, sowie einem optionalen INTEGER-Feld, `pathLenConstraint`. Für CA-Zertifikate muß das `cA`-Feld auf TRUE gesetzt werden, für andere auf FALSE. Laut RFC 2459 sollte Basic Constraints in Nicht-CA-Zertifikaten nicht verwendet werden, also auch nicht, wenn die Extension FALSE markiert ist. `pathLenConstraint` ist nur sinnvoll in CA-Zertifikaten und gibt an, wieviele CA-Ebenen unterhalb des CA-Zertifikats maximal zulässig sind. Ein Wert 0 bedeutet hierbei, diese CA gibt nur Anwendungs- bzw. Benutzerzertifikate heraus. Basic Constraints sollte immer als Critical markiert werden.

Laut *Stephen N. Henson* <http://www.drh-consultancy.demon.co.uk/caornot.html> ist die Basic Constraints Extension unbedingt erforderlich in CA-Zertifikaten, die S/MIME Benutzer zertifizieren. Andernfalls wird die S/MIME-Mail beim Empfänger aufgrund eines ungültigen CA-Zertifikats zurückgewiesen.

Sowohl der Netscape- als auch der Microsoft-Browser interpretieren die Extension. (Henson hat allerdings *die Erfahrung* <http://www.drh-consultancy.demon.co.uk/ca-fix.html> gemacht, daß „Microsoft Outlook Express 98" grundsätzlich Schwierigkeiten mit Critical markierten Extension hat: *„Unfortunately Microsoft Outlook 98 chokes on critical extensions..."*.) Outlook Express 5 scheint diese Probleme nicht mehr zu haben.

(Für genauere Angaben siehe *RFC 2459* sowie die Beispiel-Konfigurationsdatei im Anhang.)

In der Konfigurationsdatei:

```
basicConstraints = [critical,] CA:<TRUE|FALSE>[, pathlen:n]
```

## 4.3 Key Usage

Key Usage steuert den Verwendungszweck des zu einem Zertifikat gehörenden Schlüssels: z.B. darf ein Schlüssel nur zum Signieren von CRL's, nur zur Daten-Verschlüsselung oder nur zum Unterschreiben verwendet werden. Laut Netscape Dokumentation vom 13.08.97, *Netscape Certificate Extensions - Communicator 4.0 Version* <http://home.netscape.com/eng/security/comm4-cert-exts.html>, wird Key Usage vom Netscape Browser (NSC) zur Beschränkung der Verwendung von Zertifikaten ausgewertet. Allerdings nur, wenn Key Usage als **Critical** (s.u. (4.1)) markiert ist. Liegen andererseits mehrere Zertifikate vor (z.B. eines zum Signieren, eines zum Verschlüsseln), bestimmt Key Usage (Critical oder nicht), welches Zertifikat vom Browser verwendet wird.

Laut Microsoft-Dokumentation *Structuring X.509 Certificates for Use with Microsoft Products* <http://www.microsoft.com/security/ca/structuring.htm> wertet der MS Internet Explorer (MSIE) Key Usage aus, egal ob Critical oder nicht. Das deckt sich aber nicht ganz mit den gemachten Erfahrungen (siehe unten).

Üblicherweise wird Key Usage eingesetzt, um die Verwendung des Public Keys (und somit auch des Private Keys), an den diese Extension durch die Zertifizierung gebunden wird, zu beschränken. *Das funktioniert natürlich nur, wenn die Applikation, mit der das Zertifikat verwendet wird, diese Extension auch interpretiert.* Der Netscape Browser (4.06) beispielsweise verweigert den Kontakt zu einem SSL-Server, wenn im Server-Zertifikat das Flag für Digital Signature gesetzt *und* dieses Critical markiert ist. Laut *Netscape Dokumentation* <http://home.netscape.com/eng/security/comm4-cert-exts.html> sollte dieses Flag in einem SSL-*Client*-Zertifikat gesetzt sein. Für einen SSL-Server hätte dagegen Key Encipherment gesetzt sein müssen. Der Browser läßt daher keine SSL-Verbindung zu und bricht den Verbindungswunsch mit der Meldung „The certificate is not approved for the attempted operation" ab. Dasselbe Zertifikat wurde aber vom Microsoft-Browser (Ver. 4.01) problemlos akzeptiert. In der Microsoft-Dokumentation *Structuring X.509 Certificates for Use with Microsoft Products* <http://www.microsoft.com/security/ca/structuring.htm> vom 4.12.97 steht im Abschnitt zum Thema Key Usage: „The only Microsoft Application that currently enforces KeyUsage is Microsoft Outlook."

Die nachstehende Beschreibung erfolgt in Anlehnung an die oben erwähnte Netscape-Dokumentation und den *RFC 2459*. Es stehen neun Werte für das Schlüsselwort `keyUsage` in der Konfigurationsdatei `openssl.cnf` zur Verfügung:

(Für genauere Angaben siehe *RFC 2459* sowie die Beispiel-Konfigurationsdatei im Anhang.)

In der Konfigurationsdatei:

```
keyUsage = [critical,] ein oder mehrere der obigen Bezeichner (mittlere Spalte)
```

| Bezeichnung | Wert für `keyUsage` in `openssl.cnf` | Verwendung des Public Keys |
|---|---|---|
| Decipher Only | `decipherOnly` | Ist *Key Agreement* gesetzt, darf der Public-Key innerhalb eines Schlüsselaustausches zur Entschlüsselung von Daten verwendet werden. Andernfalls undefiniert. |
| Encipher Only | `encipherOnly` | Ist *Key Agreement* gesetzt, darf der Public-Key innerhalb eines Schlüsselaustausches zur Verschlüsselung von Daten verwendet werden. Andernfalls undefiniert. |
| CRL Signing | `cRLSign` | Public Key kann verwendet werden, um CRLs zu verifizieren. |
| Key Cert Sign | `keyCertSign` | Public Key kann verwendet werden, um Zertifikate zu verifizieren. |
| Key Agreement | `keyAgreement` | Zur Verwendung beim Schlüsselaustausch. |
| Data Encipherment | `dataEncipherment` | Zur Verschlüsselung von „normalen“ Daten, also keinen Schlüsseln. |
| Key Encipherment | `keyEncipherment` | Public Key wird zum Schlüsselmanagement verwendet. |
| Non Repudiation | `nonRepudiation` | Key zur Prüfung von „bewußten“ Signaturen (außer CRLs und bei Zertifikaten). |
| Digital Signature | `digitalSignature` | Key zur Prüfung von „automatisierten“ Signaturen (außer bei CRLs und bei Zertifikaten). |

## 4.4 Extended Key Usage

Extended Key Usage kann ergänzend oder anstelle von Key Usage verwendet werden. Die Extension beschränkt analog Key Usage die Verwendung des zertifizierten Public-Keys. Beispielsweise könnte die Verwendung von CA-Zertifikaten, welche die entsprechende Basic Constraints und Key Usage Extension enthalten, feiner unterschieden werden. Extended Key Usage könnte dazu auf `serverAuth` gesetzt sein, so daß der CA-Schlüssel lediglich zum Überprüfen von SSL-Server- aber nicht von SSL-Client-Zertifikaten dient.

Auch Extended Key Usage kann `critical` gesetzt sein und es gilt das im Abschnitt Critical Bit (4.1) gesagte.

Jede Organisation kann eigene Werte für die Extension Extended Key Usage registrieren lassen. Unter anderem Netscape und Microsoft haben das getan.

OpenSSL kennt die in der Tabelle aufgeführten symbolischen Werte für das Schlüsselwort `extendedKeyUsage`. Es können aber auch andere Werte für die Extension durch Angabe der entsprechenden OID gesetzt werden.

In der Microsoft-Dokumentation *Structuring X.509 Certificates for Use with Microsoft Products* <`http://www.microsoft.com/security/ca/structuring.htm`> vom 4.12.97 steht, daß für den Einsatz von Zertifikaten im Zusammenhang mit Microsofts „Authenticode“ nur der Wert `codeSigning` für Extended Key Usage angegeben sein darf. Ebenfalls in dieser Dokumentation steht, daß die Gültigkeit von Extended Key Usage im Anwendungszertifikat nur gegeben ist, *wenn sämtliche Zertifikate der Zertifikatkette diese Extension enthalten*. Ist die Extension dagegen garnicht enthalten, sollen MS-Anwendungen das Zertifikat für jeden durch Key Usage beschriebenen Zweck als gültig interpretieren.

Auch Netscape scheint das Setzen der Extension in allen Zertifikaten der Kette zu unterstützen. Es scheint aber doch fraglich, wieviel Sinn es beispielsweise macht, in einen CA-Zertifikat Extended Key Usage auf `email` zu setzen, damit die Extension in einem Anwendungszertifikat gültig ist. Die Empfehlungen von Netscape für diese Extension können in *Installation and Deployment Guide, Appendix B* <`http://developer.netscape.com:80/docs/manuals/cms/41/dep_gide/ext.htm`> nachgelesen werden.

Zu Netscapes bzw. Microsofts SGC („Server Gated Cryptography“) ist anzumerken, daß die Verwendung dieser Werte nur im Zusammenhang mit speziellen CA-Zertifikaten sinnvoll ist. Diese CA-Zertifikate werden durch ein Flag *im Browser* als für SGC geeignet gekennzeichnet. Ein eigenes CA-Zertifikat kann nur nach Import in den Browser und anschließendem Patchen der Zertifikat-Datenbank (Netscape) mit diesem Flag

| Bezeichnung | Wert für `extKeyUsage` in `openssl.cnf` | Verwendung des Public Keys |
|---|---|---|
| RFC2459-Extensions | | |
| TLS Web Server Authentication | `serverAuth` | Authentisierung von Web-Servern durch Web-Clients |
| TLS Web Client Authentication | `clientAuth` | Authentisierung von Web-Clients durch Web-Server |
| Code Signing | `codeSigning` | Key zur Signierung von Programm-Code |
| Email Protection | `emailProtection` | Key zur Verwendung mit S/MIME-Software |
| Time Stamping | `timeStamping` | Signierung von Objekt-Hashwerten und zugehörigen vertrauenswürdigen Zeitstempeln |
| Microsoft-Extensions | | |
| Individual Code Signing | `msCodeInd` | |
| Commercial Code Signing | `msCodeCom` | |
| Trust List Signing | `msCTLSign` | |
| Server Gated Crypto | `msSGC` | Server-Zertifikat mit „Global Server ID“ |
| Encrypted File System | `msEFS` | Verschlüsselung von symmetrischen Keys zur Dateissystem-Verschlüsselung |
| Netscape-Extensions | | |
| Server Gated Crypto | `nsSGC` | Server-Zertifikat mit „Global Server ID“ |

ausgestattet werden. Genaueres steht in der Datei `README.GlobalID` des SSL-Apache-Pakets *ModSSL* `<http://www.modssl.org/>`

Eine Empfehlung für die Werte dieser Extension ist nur schwer zu geben, gerade wegen der MS-Forderung, daß die Extension in allen Zertifikaten der Kette enthalten sein muß. Am sinnvollsten scheint es zu sein, ganz auf diese Extension zu verzichten.

(Für genauere Angaben siehe *RFC 2459* sowie die Beispiel-Konfigurationsdatei im Anhang.)

In der Konfigurationsdatei:

```
keyUsage = [critical,] ein oder mehrere der obigen Bezeichner (mittlere Spalte) [, OID]
```

## 4.5 Subject Key Identifier

Die Extension Subject Key Identifier enthält den Hash-Wert des Public Keys eines Zertifikates. Dadurch kann ein zu einem Public Key gehörendes Zertifikat effizient gesucht werden. So wird die Überprüfung von Zertifikatketten und bei mehreren Anwendungszertifikaten die Auswahl des richtigen Zertifikats unterstützt. Diese Extension sollte sowohl in CA-Zertifikaten als auch in Anwendungszertifikaten enthalten sein.

(Für genauere Angaben siehe *RFC 2459* sowie die Beispiel-Konfigurationsdatei im Anhang.)

In der Konfigurationsdatei:

```
subjectKeyIdentifier = hash
```

## 4.6 Authority Key Identifier

Die Extension Authority Key Identifier besteht aus drei Feldern: `keyIdentifier`, `authorityCertIssuer` und `authorityCertSerialNumber`. Laut RFC 2459 kann ein Schlüssel mittels dieser Extension auf zwei Arten identifiziert werden: entweder durch alleiniges Setzen des `keyIdentifier`-Feldes, oder durch Setzen der anderen beiden Felder. Diese Extension unterstützt die Überprüfung von Zertifikatketten.

Microsoft empfiehlt die zweite Variante, damit eine Zertifikatkette überprüft werden kann. RFC 2459 empfiehlt die erste Variante.

(Für genauere Angaben siehe *RFC 2459* sowie die Beispiel-Konfigurationsdatei im Anhang.)

In der Konfigurationsdatei:

```
authorityKeyIdentifier = <[keyid[:always]][, issuer[:always]]>
```

Ist `keyid` gesetzt, wird Subject Key Identifier (siehe oben (4.5)) des Herausgeber-Zertifikats kopiert. Kann der Wert dieser Extension nicht kopiert werden (weil Subject Key Identifier nicht gesetzt war) und ist `keyid` auf `always` gesetzt, wird mit einer Fehlermeldung abgebrochen. Ist `keyid` nicht `always` gesetzt, werden alternativ das Issuer-Feld und die Seriennummer kopiert. Ist `issuer` auf `always` gesetzt, werden immer das Issuer-Feld und die Seriennummer kopiert.

## 4.7 Subject Alternative Name

Die Extension Subject Alternative Name kann verwendet werden, um weitere Bezeichner für ein Subject in das Zertifikat zu bringen. Es sind E-Mail-Adressen, DNS-Namen, IP-Adressen, URIs und registrierte IDs (RIDs), auch mehrfach, möglich. Diese können auch beliebig kombiniert werden.

(Für genauere Angaben siehe *RFC 2459* sowie die Beispiel-Konfigurationsdatei im Anhang.)

In der Konfigurationsdatei:

```
subjectAltName=<[email:<copy|name@mail.de>][, URL:http://my.url.here/][, RID:1.2.3.4][,
IP:1.2.3.4]>
```

## 4.8 Issuer Alternative Name

Die Extension Issuer Alternative Name ist wie Subject Alternative Name (4.7) aufgebaut.

Hier wird allerdings nicht `email:copy` sondern `issuer:copy` unterstützt. Dabei werden die Angaben vom Subject Alternative Name des *Herausgeber*-Zertifikats kopiert.

(Für genauere Angaben siehe *RFC 2459* sowie die Beispiel-Konfigurationsdatei im Anhang.)

In der Konfigurationsdatei:

```
subjectAltName=<[issuer:copy][, URL:http://my.url.here/][, RID:1.2.3.4][, IP:1.2.3.4]>
```

## 4.9 Certificate Policies

Die Extension Certificate Policies verweist auf die Policy, unter der ein Zertifikat herausgegeben wurde. Laut RFC 2459 sollte die Extension lediglich aus einer OID (*Object Identifier*) bestehen. Eine Anwendung, die Zertifikate prüft, sollte eine Liste mit den OID's akzeptierter Policies enthalten und diese gegen die Policy-OID eines Zertifikats vergleichen. Dann wird das Zertifikat entsprechend akzeptiert oder zurückgewiesen.

Wenn die OID nicht ausreichend ist (weil z.B. die eingesetzten Applikation keine OID-Listen unterstützen), beschreibt RFC 2459 die Möglichkeit einen URI anzugeben, der auf die Policy der CA verweist (*Certification Practice Statement, CPS*). Für Anwendungszertifikate und CA-Zertifikate, die an andere Organisationen (also nicht die der Herausgeber-CA) herausgegeben werden, kann zusätzlich eine *User Notice* festgelegt werden.

Die User Notice besteht aus zwei optionalen Feldern: `noticeRef` und `explicitText`. Das erste Feld `noticeRef` besteht aus dem Organisations-Namen und einer Nummer. Der Nummer ist ein Text zugeordnet und soll wieder durch eine Anwendung mit einer entsprechen Liste interpretiert werden. Findet eine Anwendung diese Nummer in ihrer Liste, soll sie den zugehörigen Text anzeigen. Das zweite Feld enthält einen, maximal 200 Zeichen umfassenden, frei gestaltbaren Text und steht direkt im Zertifikat. Eine Anwendung soll diesen dann bei Verwendung des Zertifikats anzeigen.

Microsoft-Anwendungen scheinen mit der Extension Schwierigkeiten zu haben, wie Peter Gutmann in seinem *X.509 Style Guide* <http://www.cs.auckland.ac.nz/~pgut001/pubs/x509guide.txt> beschreibt:

> *„Although various MS programs give the impression of handling certificate policies, they only have a single hardcoded policy which is the Verisign CPS. To see an example of this, create a certificate with a policy of (for example) This policy isn't worth the paper it's not written onänd view the cert using Outlook Express. What's displayed will be the Verisign CPS."*

Outlook Express 5 und der Internet Explorer 5 scheinen diese Probleme nicht mehr zu haben. Eine User Notice wird für Benutzerzertifikate korrekt angezeigt. Die URL für die CPS wird ebenfalls ausgewertet und die entsprechende Seite angezeigt.

(Für genauere Angaben siehe *RFC 2459* sowie die Beispiel-Konfigurationsdatei im Anhang.)

In der Konfigurationsdatei:

```
certificatePolicies=[ia5org,]OID[, OID, ...][,@polsect]
```

Laut OpenSSL-Dokumentation von Stephen N. Henson (im Quell-Verzeichnis des OpenSSL-Pakets unter `doc/openssl.txt`) ist die `ia5org`-Option für die Verwendung mit dem Internet Explorer erforderlich, obwohl sie nicht RFC-konform ist.

Die Option `@polsect` verweist auf einen Abschnitt in der Konfigurationsdatei, der die Werte für das CPS und indirekt für das User Notice Feld enthält:

```
[ polsect ]
policyIdentifier=OID
CPS="http://www.ca.de/policy.html"
userNotice=@notice

[ notice ]
explicitText="Nur zur Verschluesselung von E-Mail (S/MIME)"
organisation="CA Org."
noticeNumbers=4, 2
```

## 4.10 CRL Distributions Points

Die Extension CRL Distributions Points legt fest, wo eine Widerrufliste (*CRL*) der Herausgeber-CA abgerufen werden kann. Grundsätzlich sind laut RFC 2459 alle Optionen, die bei Subject Alternative Name (4.7)

verwendet werden können, auch hier einsetzbar. OpenSSL unterstützt allerdings bisher nur die Verwendung eines URIs. Dieser muß aus der absoluten Adresse der CRL bestehen.

(Für genauere Angaben siehe *RFC 2459* sowie die Beispiel-Konfigurationsdatei im Anhang.)

In der Konfigurationsdatei:

```
crlDistributionPoint=URI:http://www.ca.de/ca1.crl[, URI:http://www.ca.de/ca2.crl ... ]
```

## 4.11 Netscape Certificate Extensions

Zu den *Netscape Certificate Extensions* <http://home.netscape.com/eng/security/certs.html> ist anzumerken, daß diese nicht Critical gesetzt werden sollten. Ein SSL-Server-Zertifikat, in dem beispielsweise das Netscape-SSL-Server-Bit (`nsCertType=critical,server`) Critical gesetzt ist, wird von einem Microsoft-Browser (der diese Extension nicht kennt) zurückgewiesen werden. Ein solches Server-Zertifikat würde also eine SSL-Verbindung von Microsoft-Clients zum Server ausschließen.

Folgende Netscape-Erweiterungen werden von OpenSSL-0.9.5-dev unterstützt (die Schlüsselwörter für die OpenSSL-Konfigurationsdatei stehen in Klammern):

- netscape-cert-type (`nsCertType`)
- netscape-base-url (`nsBaseUrl`)
- netscape-revocation-url (`nsRevocationUrl`)
- netscape-ca-revocation-url (`nsCaRevocationUrl`)
- netscape-cert-renewal-url (`nsRenewalUrl`)
- netscape-ca-policy-url (`nsCaPolicyUrl`)
- netscape-ssl-server-name (`nsSslServerName`)
- netscape-comment (`nsComment`)

Wichtig ist die Erweiterung Netscape Cert Type, um für die mit OpenSSL erzeugten Zertifikate den Verwendungszweck (zumindest für Netscape-Browser) zu beeinflussen. Der Internet-Explorer scheint die Netscape-Extensions zu ignorieren. Für die Erweiterung Netscape Cert Type stehen die folgenden acht Werte für das Schlüsselwort `nsCertType` in der Konfigurationsdatei zur Verfügung:

| Wert für `nsCertType` in `openssl.cnf` | Bedeutung |
|---|---|
| `objCA` | Ein CA-Zertifikat um Zertifikate zum Signieren von Objekten (Java-Applets etc.) herauszugeben. |
| `emailCA` | Ein CA-Zertifikat um E-Mail-Benutzer zu zertifizieren |
| `sslCA` | Ein CA-Zertifikat um Server oder Clients zu zertifizieren |
| `reserved` | Reserviert für zukünftige Nutzung |
| `objsign` | Zertifikat um Objekte (Java-Applets etc.) zu signieren |
| `email` | Ein E-Mail Benutzer-Zertifikat |
| `server` | Ein SSL-Server-Zertifikat |
| `client` | Ein SSL-Client-Zertifikat |

Es sind auch Kombinationen der einzelnen Bezeichnungen möglich, so steht z.B. `nsCertType=objCA, emailCA, sslCA` für ein CA-Zertifikat, mit dem Zertifikate für Objekt-Signierung, S/MIME-Benutzer und SSL-Server/-Clients herausgegeben werden können. Die Bedeutung der anderen Attribute kann dem Anhang `openssl.cnf (A)` entnommen werden.

# 5 OpenSSL-Konfigurationsdatei, openssl.cnf

Eine Beispiel-Konfigurationsdatei findet sich im Anhang **openssl.cnf (A)**.

Die Belegung von Aufruf-Optionen der OpenSSL-Applikationen wird weitestgehend durch die Konfigurationsdatei **$(SSLETC)/openssl.cnf** bestimmt. Sie ist ähnlich aufgebaut wie eine Windoze-Ini-Datei. Einzelne Abschnitte sind gekennzeichnet durch Bezeichner der Form **[ uvwxyz ]**. Innerhalb dieser Abschnitte können Variablen vereinbart werden. Dabei ist es auch möglich, Umgebungsvariablen zu überschreiben, z.B. mit

```
ENV::PATH = /usr/local/bin:$PATH
```

Außerdem können Werte von gesetzten Umgebungsvariablen einzelnen Variablen der Konfigurationsdateien zugeordnet werden, z.B. mit

```
ca_default = $ENV::ENV_CA_DEFAULT
```

Enthält die Umgebungsvariable **ENV_CA_DEFAULT** den Wert **Server_CA**, wird bei der nächsten Zertifizierung durch die Applikation **ca** der Abschnitt **ca_default** gewählt. Analog könnte eine Umgebungsvariable **ENV_EXT**, die die Bezeichnung eines Abschnitts mit Zertifikat-Erweiterungen *Extensions* enthält, gesetzt werden:

```
x509_extensions = $ENV::ENV_EXT
```

Die Datei gliedert sich grob in zwei Abschnitte: **[ ca ]**, in dem Voreinstellungen für die Erzeugung eines Zertifikats vorgenommen werden, und entsprechend **[ req ]** für die Erzeugung eines „Zertifizierungswunsches“ (*Request*). Auf diese voreingestellten Abschnitte wird zugegriffen, wenn **openssl** mit dem Parameter **ca** bzw. **req** aufgerufen wird. (Anm.: **openssl req** *Parameter...* **erzeugt** einen Request, **openssl ca** *Parameter...* **signiert** einen Request.)

Sowohl **ca** als auch **req** bieten die Möglichkeit, über die Option **-config** die zu verwendende Konfigurationsdatei explizit anzugeben. So könnte jeweils eine Konfigurationsdatei speziell für Netscape-Browser und eine andere für Microsoft-Browser gestaltet sein.

Es ist auch möglich, innerhalb einer Konfigurationsdatei mehrere CA-Abschnitte zu definieren, je nachdem, welche Art Request signiert werden soll. Dadurch kann innerhalb einer Konfigurationsdatei eine CA-Konfiguration zur Server-Zertifizierung (z.B. **[ Server_CA ]**), eine Konfiguration zur Client-Zertifizierung (z.B. **[ Client_CA ]**) und eine Konfiguration zur *S/MIME* **<http://www.rsa.com/rsa/S-MIME/>**-Zertifizierung (z.B. **[ SMIME_CA ]**) verwaltet werden. Bei Aufruf von **openssl** mit dem Parameter **ca** kann dann mit der Option **-name Client_CA** z.B. auf eine Client-CA-Konfiguration zugegriffen werden. Alternativ kann die Auswahl des Konfigurations-Abschnitts durch die oben erwähnte Umgebungsvariable **ENV_CA_DEFAULT** erfolgen. (Anm.: Statt der im letzten Absatz erwähnten zwei speziellen Konfigurationsdateien kann das gleiche auch über entsprechende Konfigurationsabschnitte innerhalb einer Datei erreicht werden.)

Sollen verschiedene Zertifikattypen durch eine CA herausgegeben werden, können unterschiedlich benannte Abschnitte, die die jeweiligen Extensions enthalten, in der Konfigurations-Datei angelegt werden. Der gewünschte Abschnitt wird dann über die Umgebungsvariable **ENV_EXT** ausgewählt, wenn in der Konfigurationsdatei **x509_extensions = $ENV::ENV_EXT** gesetzt ist. (Anm.: Das gleiche kann auch über die oben erwähnten speziellen Konfigurationsdateien erreicht werden.)

Innerhalb von CA-Abschnitten werden drei Schlüsselwörter erkannt, die auf weitere Abschnitte in einer Konfigurationsdatei verweisen. Die Schlüsselwörter lauten:

- **x509_extensions**

- `crl_extensions`
- `policy`

`x509_extensions` verweist auf einen Abschnitt, in dem Extensions für neue Zertifikate festgelegt sind. In diesem Abschnitt könnten die Extensions für ein Benutzer- oder ein CA-Zertifikat festgelegt werden.

`crl_extensions` verweist auf einen entsprechenden Abschnitt für Extensions, die in eine Zertifikat-Widerrufliste (*certificate revocation list, CRL*) gebracht werden sollen.

`policy` schließlich weist auf einen Abschnitt, in dem festgelegt wird, inwieweit der Name in einem zu signierenden Request mit dem des CA-Zertifikats übereinstimmen muß. Beispielsweise kann festgelegt werden, daß die Angaben zum Land übereinstimmen müssen.

Im `req`-Abschnitt werden vier Schlüsselwörter erkannt:

- `distinguished_name`
- `attributes`
- `x509_extensions`
- `req_extensions`

`distinguished_name` weist auf einen Abschnitt, in dem festgelegt wird, welche Namensfelder bei der Erzeugung des Requests abgefragt werden, sowie deren Default-Werte.

Über `attributes` werden optionale Felder festgelegt, die zusätzlich in den Request gebracht werden können. Diese dienen zum Widerruf eines Zertifikats bei Verlust des Private-Key. Genaueres steht im Abschnitt Erzeugen von Requests (6.2).

Auch im `req`-Abschnitt gibt es einen Bereich, auf den ein Schlüsselwort `x509_extensions` verweist. Im Unterschied zu `x509_extensions` im `ca`-Bereich, werden hier die Extensions festgelegt, die bei Erzeugung eines *selbstsignierten* Zertifikats (*Wurzel-* oder *Root-Zertifikat*) in ein solches Zertifikat gebracht werden.

Im vierten Bereich, auf den `req_extensions` weist, werden Extensions festgelegt, die in einen *Request* gebracht werden. Üblicherweise werden Extensions allerdings durch eine CA beim Signieren eines Requests in das Zertifikat gebracht. Es ist aber auch denkbar, daß der Zertifikatnehmer bei der Erzeugung eines Requests festlegt, welche Extensions sein Zertifikat enthalten soll. Eine CA könnte dann diese Extensions in das Zertifikat übernehmen. Es sollte unbedingt mit der CA abgesprochen werden, ob sie Requests mit Extensions signiert.

Ein ausführliches Beispiel einer Konfigurationsdatei mit mehreren Abschnitten findet sich im Anhang `openssl.cnf (A)`

# 6 Erzeugen von Requests und Zertifikaten

Die folgenden Abschnitte beschreiben die Erzeugung eines selbstsignierten Zertifikats, das sogenannte *Root-Zertifikat*, sowie Erzeugung und Signierung eines Requests. Dabei sollte beachtet werden, daß die Gültigkeitsdauer eines neuen Zertifikats vollständig innerhalb des Gültigkeitsbereiches des Zertifikats der signierenden CA liegt. MS-Produkte weisen sonst möglicherweise dieses neue Zertifikat als ungültig zurück. Weiter ist es empfehlenswert, nach Herausgabe eines CA-Zertifikats einen Tag zu warten, bevor der zugehörige Private-Key zur Zertifizierung eingesetzt wird.

Netscape- und Microsoft-Browser mit einer Versionsnummer kleiner als 4.0 *können nicht mit Schlüssellängen größer als 1024 Bit umgehen*. Das gilt sowohl für CA- als auch für Anwendungs-Zertifikate. Die Browser

zeigen eine irreführende Meldung an, daß das Server-Zertifikat eine ungültige Signatur hat oder grundsätzlich fehlerhaft ist, obwohl das Zertifikat technisch korrekt ist.

Im Verzeichnis `$(SSLETC)/misc` gibt es eine Perl-Skript, `CA.pl`. Mit dem Skript können ebenfalls Zertifikate und Requests erzeugt werden. Das Skript soll den Einstieg in die Handhabung von Zertifikaten mit OpenSSL erleichtern, indem komplexe Kommandos und die Konfiguration gekapselt werden. Das Skript ist dokumentiert, muß aber möglicherweise noch angepaßt werden. Auf den Einsatz des Skriptes wird hier nicht weiter eingegangen.

## 6.1 Erzeugen eines Root-CA-Zertifikats

Die im obigen Abschnitt (4) aufgeführten Erweiterungen werden üblichererweise beim Zertifizieren eines Requests mit OpenSSL durch eine CA in das Zertifikat gebracht. In der Konfigurationsdatei (siehe `openssl.cnf (A)`) gibt es einen Abschnitt `[ v3_ca ]`, in dem festgelegt werden kann (und sollte!), welche Zertifikat-Erweiterungen bei Erzeugung eines (selbstsignierten) **Root-Zertifikats** gesetzt werden.

Im folgenden ein Beispiel zur Erzeugung eines Root-CA-Zertifikats:

1. Editieren von `openssl.cnf`, Setzen der Zertifikat-Erweiterungen im Abschnitt `[ v3_ca ]`. Um z.B. ein Root-CA-Zertifikat zu erzeugen, mit dem SSL-CA-, S/MIME-CA- und Object-Signing-CA-Zertifikate herausgegeben werden können, sollte die Netscape Certificate Extension folgenderweise gesetzt sein:

   ```
   nsCertType = sslCA, emailCA, objCA
   ```

2. Mögliche Werte für die anderen Extensions:

   ```
   basicConstraints       = critical, CA:TRUE
   keyUsage               = cRLSign, keyCertSign
   subjectKeyIdentifier   = hash
   authorityKeyIdentifier = keyid, issuer:always
   subjectAltName         = email:copy
   issuerAltName          = issuer:copy
   crlDistributionPoints  = URI:http://crlserver.domain.de/CA.crl
   ```

   Die unter 1. und 2. aufgeführten Einträge müssen in der Konfigurationsdatei, in dem beim Schlüsselwort `x509_extensions` genannten Bereich gemacht werden, also z.B. im Bereich `[ v3_ca ]`. Das Schlüsselwort `x509_extensions` kann sowohl im `ca`-Abschnitt als auch im `req`-Abschnitt stehen. Für das selbstsignierte Zertifikat muß `x509_extensions` im Bereich `[ req ]` der Konfigurationsdatei auf `v3_ca` gesetzt werden.

3. Initialisieren des Zufallszahlen-Generator-Status. Dieser Status ist eine Datei, die durch jeden Zugriff verändert wird. Die Datei kann z.B. als `/usr/local/etc/ssl/private/.rand` engelegt werden. Es sollte die Umgebungsvariable `$RANDFILE` auf diese Datei gesetzt werden. Initialisiert wird der Status dann durch folgenden Befehl:

   ```
   cp ~/.pgp/randseed.bin $RANDFILE
   ```

4. Erzeugen eines 2048-Bit-Schlüssels:

   Vor Aufruf dieses Befehls muß die Umgebungsvariable `$RANDFILE` gesetzt sein, sonst findet das folgende Kommando die Datei mit dem Zufallszahlen-Status nicht, und ein einkompilierter Default wird wirksam.

```
openssl genrsa -des3 -out $(SSLETC)/private/CAkey.pem -rand Zufallsdaten 2048
```

*Achtung:*
Ohne die Option `-des3` (oder `-des`, `-idea`) wird der Schlüssel unverschlüsselt abgespeichert!

5. Erzeugen des selbstsignierten Root-CA-Zertifikats:

```
openssl req -new -x509 -days 730 -key $(SSLETC)/private/CAkey.pem -out
$(SSLETC)/private/CAcert.pem
```

*Achtung:*
Über die Option `-days` wird die Gültigkeitsdauer des Root-Zertifikats in Tagen festgelegt. Beginn des Zeitraums ist der Zeitpunkt der Erzeugung des Zertifikats.

6. Anzeigen des erzeugten Zertifikats:

```
openssl x509 -in ./CAcert.pem -text | more
```

Das Zertifikat muß jetzt noch in das Verzeichnis `$(SSLETC)/certs` als `00.pem` kopiert werden. Der Name ergibt sich aus der hexadezimalen Seriennummer des Zertifikats (hier 00) und dem Anhang `.pem`. Anschließend sollte das Zertifikat noch über seinen Hash-Wert verlinkt werden (siehe Anmerkung unten (6.1)).

- `cp $(SSLETC)/private/CAcert.pem $(SSLETC)/certs/00.pem`
- `cd $(SSLETC)/certs`
- `ln -s 00.pem ‘openssl x509 -hash -noout -in 00.pem‘.0`

Alternativ zu dem Link-Befehl kann auch das Shell-Skript `c_rehash` in `$(SSLETC)/bin` aufgerufen werden. Das Skript erzeugt für alle im Verzeichnis `certs` gefundenen Zertifikate die nötigen Links. Der Hash-Wert besteht aus 64 Bit und wird aus den Angaben des Subject-Feldes (also Distinguished Name und evtl. Email-Adresse des Herausgebers) des Zertifikats gebildet.

*Anmerkung:*
Alle neuen Zertifikate sollten über ihren Hash-Wert verlinkt werden. Das ist deshalb von Bedeutung, weil die Suche nach Zertifikaten und der damit verbundene Vergleich *ausschließlich* über die Hash-Werte der Zertifikate erfolgt.

## 6.2 Erzeugen eines Certificate Requests

Zur Erzeugung eines beliebigen Requests (CA, Server, Client) muß zunächst ein Schlüsselpaar generiert werden. Mit folgendem Befehl wird ein 1024 Bit Schlüssel erzeugt:

```
openssl genrsa -des3 -out MyKey.pem -rand file1:file2:... 1024
```

Vorher sollte allerdings der Zufallszahlen-Status initialisiert worden sein. Die nach der Option `-rand` angegebenen Dateien dienen als zusätzliche Zufallsdaten für die Schlüsselerzeugung. Abhängig vom später verwendeten Browser ist die Schlüssellänge zu beachten. Für den MS-Internet-Explorer in der *internationalen (Export-)Version* ist die Schlüssellänge für ein Benutzerzertifikat grundsätzlich auf 512 Bit begrenzt. Die `1024` in obigen Kommando muß dann durch `512` ersetzt werden. Für die Netscape Navigator/Communicator (NSC)Export-Version gilt im Prinzip die gleiche Beschränkung. Die Schlüssellänge kann aber bis auf 2048 Bit erhöht werden, indem das Zertifikat extern (z.B. mit `openssl`) erzeugt und anschließend als `.p12`-Datei (siehe Hensons *PKCS#12-Seite* <`http://www.rsa.com/rsalabs/pubs/PKCS/html/pkcs-12.html`>) in den

NSC importiert wird (siehe PFX (9.1) bzw. PKCS12 (8.2)). *Da die USA die Export-Beschränken für Kryptosoftware gelockert haben, sollten demnächst Browser-Versionen zur Verfügung stehen, die diese Beschränkung der Schlüssellänge nicht mehr haben.*

Die Schlüsselerzeugung hätte bei dem folgenden Request-Kommando gleichzeitig mit der Request-Erzeugung durch die Option `-newkey` veranlaßt werden können; diese Variante bietet aber keine Möglichkeit, zusätzliche Zufallsdaten anzugeben.

Die Erzeugung des Requests erfolgt durch folgenden Befehl:

```
openssl req -new -key MyKey.pem -out MyReq.pem
```

Nach Eingabe des Befehls müssen folgende Angaben gemacht werden (beispielhafte Eingaben sind in doppelten Anführungsstrichen):

```
Using configuration from /usr/local/etc/ssl/openssl.cnf
Enter PEM pass phrase:  "passphrase"
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]: "DE"
State or Province Name (full name) [Some-State]: "Schleswig-Holstein"
Locality Name (eg, city) []: "Kiel"
Organization Name (eg, company) [Internet Widgits Pty Ltd]: "Universitaet Kiel"
Organizational Unit Name (eg, section) []: "Studis"
Common Name (eg, YOUR name) []: "Fred Neumann"
Email Address []: "neumann@inf.uni-kiel.de"

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

(Für den genauen Inhalt muß die zertifizierende CA bzw. deren Policy konsultiert werden.)

Die beiden letzten Angaben tauchen als Klartext im Attribut-Bereich des Requests auf. Soll später ein Zertifikat zurückgerufen werden, kann sich der Inhaber gegenüber der CA, auch bei Verlust des Private Keys, durch Angabe dieser Felder als Inhaber „ausweisen“. Die Verwaltung dieser Angaben kann von der CA durchgeführt werden.

Je nach Verwendungszweck des späteren Zertifikats muß bei den Angaben folgendes beachtet werden:

- Verwendung als S/MIME-Zertifikat:

  Die E-Mail-Adresse muß vorhanden sein oder sie wird über die Extension Subject Alternative Name (4.7) gesetzt (Aufgabe der CA).

- Verwendung als SSL-Server-Zertifikat:

  Als CommonName muß der Server-Name angegeben werden, z.B. `www.site.com`. Ebenfalls sinnvoll ist es, den Server-Namen zusätzlich über die Extension `nsSslServerName` zu setzen (Aufgabe der CA).

- Verwendung als CA-Zertifikat:

  Keine Vorgaben. Sinnvoll wäre die Angabe der E-Mail-Adresse des CA-Administrators.

Es scheint Probleme mit dem Navigator und dem Internet-Explorer zu geben, wenn einige Zeichen wie „_“ oder „&“ im DN auftauchen. Stephen N. Henson empfiehlt daher, nur Zeichen vom Typ *Printable String* (mit Ausnahme der E-Mail-Adresse) zu verwenden. Die so zulässige Menge an Zeichen umfaßt die folgenden:

```
A-Z a-z 0-9 ' ( ) + , - . / : = ? leerzeichen
```

Der erzeugte Request (also nur die Datei *MyReq.pem*, nicht die Datei mit dem Schlüsselpaar) kann dann auf Diskette kopiert und einer CA zum Signieren (also: Zertifizieren, siehe Abschnitt Signieren (6.3)) vorgelegt werden. Nach der Zertifizierung kann dann das Zertifikat zusammen mit dem privaten Schlüssel als `.p12`-Datei in den jeweiligen Browser importiert werden (siehe PFX (9.1) und PKCS12 (8.2)).

## 6.3 Signieren eines Certificate Requests

Das Signieren eines einzelnen Requests erfolgt durch folgenden Befehl:

```
openssl ca -name ca_name -keyfile CAkey.pem -in MyReq.pem -out MyCert.pem -outdir
$(SSLETC)/certs
```

Der Wert von *ca_name* steht hier für den gewünschten Abschnitt der Konfigurationsdatei, also z.B. `Client_CA`. Es ist auch möglich, Gültigkeitsbeginn und -ende für das Zertifikat festzulegen. Dazu muß das obige Kommando um die Optionen `-startdate` und `-enddate` erweitert werden. Als Parameter wird bei den Optionen ein UTC-Zeitstempel der Form `YYMMDDHHMMSSZ` angegeben. Wegen der UTC-Zeit muß bei der Angabe der Stunden während der Winterzeit eine Stunde, und während der Sommerzeit zwei Stunden zur lokalen Zeit (Deutschland), bzw. zu dem gewünschten Zeitpunkt, hinzugefügt werden.

Nach Eingabe des Befehls kommt eine Meldung folgender Art (Eingaben stehen in doppelten Anführungsstrichen):

```
Using configuration from /usr/local/etc/ssl/openssl.cnf
Enter PEM pass phrase: "passphrase"
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName           :PRINTABLE:'DE'
stateOrProvinceName   :PRINTABLE:'Schleswig-Holstein'
localityName          :PRINTABLE:'Kiel'
organizationName      :PRINTABLE:'Universitaet Kiel'
organizationalUnitName:PRINTABLE:'Studis'
commonName            :PRINTABLE:'Fred Neumann'
emailAddress          :IA5STRING:'neumann@inf.uni-kiel.de'
Certificate is to be certified until Feb 12 12:10:12 2001 GMT (365 days)
Sign the certificate? [y/n]: "y"

1 out of 1 certificate requests certified, commit? [y/n] "y"
Write out database with 1 new entries
Data Base Updated
```

Findet sich die Konfigurationsdatei (KD) *openssl.cnf* nicht in dem bei der Konfiguration des OpenSSL-Pakets angegebenen SSL-Verzeichnis, kann die Angabe im `ca`-Kommando mittels `-config` */pfad/Konfigname* erfolgen. Oder es wird die Umgebungsvariable `OPENSSL_CONF` auf */pfad/Konfigname* gesetzt. Wenn die Datei mit dem CA-Key in der KD angegeben ist, kann die Angabe bei `-keyfile` ebenfalls wegfallen. Mit der Option `-name` kann ein spezieller Eintrag in der KD, z.B. der Abschnitt `Server_CA`, abgearbeitet werden (siehe Anhang `openssl.cnf (A)`). Durch das Signieren werden möglicherweise einige Extensions in das Zertifikat gebracht. Vor dem Signieren eines Requests, muß unbedingt sichergestellt sein, daß die richtigen Extensions gesetzt sind bzw. der richtige Abschnitt in der KD gewählt wird (siehe Zertifizieren (6.3) und Anhang `openssl.cnf (A)`).

Für ein S/MIME-Zertifikat könnte Key Usage (`keyUsage`) auf *Digital Signature* und *Data Encipherment* gesetzt werden, sowie Netscape Cert Type (`nsCertType`) auf *email*.

Sind in der KD keine anderen Angaben gemacht worden, befindet sich das neu erstellte Zertifikat in der bei `-out` angegebenen Datei. Außerdem wird im bei `-outdir` angegebenen Verzeichnis `$(SSLETC)/certs` eine Kopie abgelegt. Der Name der Kopie setzt sich aus der Seriennummer des Zertifikats und der Endung `.pem` zusammen, also z.B. `0a.pem`. Die Seriennummer des gerade herausgegebenen Zertifikats findet sich im Zertifikat selber (`openssl x509 -in myCert.pem -noout -serial`) oder in der Datei `$(SSLETC)/serial.old`. Das neue Zertifikat muß noch über den Hash-Wert verlinkt werden. Dazu wird in das Verzeichnis `$(SSLETC)/certs` gewechselt und das folgende Kommando gegeben:

```
ln -s 0a.pem `openssl x509 -in 0a.pem -hash -noout`.0
```

# 7 Certificate Revocation List (CRL)

CRLs sind Listen, die Seriennummer und Zeitpunkt des Rückrufs eines Zertifikats enthalten, sowie den DN der CA. Sie werden von der CA signiert und in regelmäßigen Abständen veröffentlicht.

Auch CRLs können Extensions enthalten. Sie werden beim Signieren einer CRL durch eine CA in die CRL gebracht. OpenSSL unterstützt bisher die Extensions *CRL Authority Key Identifier (7.3)* und *CRL Issuer Alternative Name (7.4)*. Die Verwendung dieser Extensions zeichnet (u.a.) eine X.509*v2*-CRL aus. Werden diese Extensions nicht verwendet (durch Löschen oder Auskommentieren des Schlüsselworts `crl_extensions` in `openssl.cnf`), erzeugt obiges Kommando eine X.509*v1*-CRL. Das ist deshalb von Bedeutung, weil *Netscape Browser mit v2 CRLs (noch) nicht umgehen können*. Beim Laden einer CRL gibt der Browser die folgende Meldung aus: „*The certificate revocation list you are trying to load has an invalid format*“.

## 7.1 Aufbau der Indexdatei `index.txt`

In der Datei `index.txt` ist eine Art text-basierte Datenbank, in der die herausgegebenen Zertifikate registriert werden. Die Datei ist von großer Bedeutung für die Verwaltung der Zertifikate. Die `ca`-Applikation z.B. prüft über die letzte Spalte der Datei, ob ein DN schon vergeben wurde.

Beispieleintrag eines Zertifikats in `index.txt`:

```
V    981210145000Z   "leer"   01   unknown   /C=DE/O=Uni/OU=Inf/CN=TestCA/Email= \
                                                              testca@uni.de
```

Die Indexdatei besteht aus sechs Spalten, jeweils getrennt durch einen Tabulator (nicht durch Leerzeichen). Die Einträge in den einzelnen Spalten haben die folgende Bedeutung:

1. Status des Zertifikats. Einer der Buchstaben R (revoked), E (expired) oder V (valid).

2. Ablaufzeitpunkt des Zertifikats. Format ist YYMMDDHHMMSS*Z* in UTC-Zeit
3. Ist in obiger Beispielzeile leer. Die Spalte kann aber ebenfalls einen Ablaufzeitpunkt YYMMDDHHMMSS*Z* in UTC-Zeit enthalten. Ist hier ein Zeitstempel eingetragen, entspricht er dem Widerrufszeitpunkt des Zertifikats. Dann muß in der ersten Spalte (statt V) ein R stehen. Für ein gültiges Zertifikat ist diese Spalte *leer*.
4. Hexadezimale Seriennummer des Zertifikats.
5. Wo das Zertifikat zu finden ist. Wird derzeit immer mit „unknown“ besetzt.
6. Der Name des Zertifikatinhabers. Üblicherweise der Distinguished Name und die E-Mail-Adresse.

Grundsätzlich ist es möglich, über Veränderung der Indexdatei eine irrtümliche Zertifizierung „rückgängig“ zu machen. Das wird erreicht, indem die letzte Zeile gelöscht, und der Zähler in der Datei `serial` um Eins erniedrigt wird. Das gilt allerdings nur für das *zuletzt* herausgegebene Zertifikat, und auch nur, wenn es noch nicht veröffentlicht wurde. Grundsätzlich ist für einen ernsthaften CA-Betrieb von solchen Veränderungen der Indexdatei abzuraten. Der bessere Weg wäre der Widerruf des betreffenden Zertifikats.

## 7.2 Widerruf eines Zertifikats

In Abhängigkeit vom Status der Zertifikate, die in der Indexdatei `$(SSLETC)/index.txt` durch Gültigkeit, Seriennummer und Distinguished Name (DN) repräsentiert werden, kann mit dem folgenden Kommando eine Certificate Revocation List (CRL) erzeugt werden. Anschließend muß die CRL noch in das DER-Format gewandelt werden, da die viele Browser nur mit diesem Format umgehen können.

```
openssl ca -gencrl -out $(SSLETC)/crl/crl.pem

openssl crl -in $(SSLETC)/crl/crl.pem -outform der -out $(SSLETC)/crl/crl.der
```

Vorher müssen die gewünschten Zertifikate zurückgerufen werden. Dazu wird folgendes Kommando verwendet:

```
openssl ca -revoke cert.pem
```

Die von dem Kommando durchgeführten Änderungen der Indexdatei sind die selben, die im folgenden Verfahren beschrieben werden.

Alternativ kann die Indexdatei mit einem Texteditor geändert werden. Soll ein Zertifikat widerrufen werden, kann mit einem Editor die Indexdatei (von Hand oder per Skript) geändert werden. In der ersten Spalte muß das `V` durch ein `R` ersetzt und in der dritten (bisher leeren) Spalte muß der Widerrufszeitpunkt eingetragen werden. Wegen der UTC-Zeit muß bei der Angabe der Stunden während der Winterzeit eine Stunde, und während der Sommerzeit zwei Stunden, zur lokalen Zeit (Deutschland) bzw. zu dem gewünschten Zeitpunkt, hinzugefügt werden.

Dann kann mit dem obigen Kommando `ca -gencrl` eine CRL mit dem widerrufenen Zertifikat erzeugt werden.

## 7.3 CRL Authority Key Identifier

Diese Extension besteht aus drei Feldern: `keyIdentifier`, `authorityCertIssuer` und `authorityCertSerialNumber`. Laut RFC 2459 kann der Schlüssel mittels dieser Extension auf zwei

Arten identifiziert werden: entweder durch alleiniges Setzen des keyIdentifier-Feldes oder durch Setzen der beiden anderen Felder. Die Extension dient dazu, Zertifikate der Herausgeber-CA zu identifizieren.

RFC 2459 empfiehlt die erste Variante.

(Für genauere Angaben siehe *RFC 2459* sowie die Beispiel-Konfigurationsdatei weiter unten.)

In der Konfigurationsdatei:

```
authorityKeyIdentifier = <[keyid[:always]][,issuer[:always]]>
```

### 7.4 CRL Issuer Alternative Name

Diese Extension ist wie Subject Alternative Name (4.7) aufgebaut und dient dazu, zusätzliche Informationen zum Herausgeber in die CRL zu bringen.

(Für genauere Angaben siehe *RFC 2459* sowie die Beispiel-Konfigurationsdatei weiter unten.)

In der Konfigurationsdatei:

```
subjectAltName=<[issuer:copy][, URL:http://my.url.here/][, RID:1.2.3.4][, IP:1.2.3.4]>
```

## 8 Testbericht und Anmerkungen

### 8.1 OpenSSL

*Die folgenden Angaben beziehen sich auf eine Entwickler-Version. Es ist daher möglich, daß die geschilderten Erfahrungen nicht mehr für die nächste Anwender-Version von OpenSSL gelten.*

Die OpenSSL-Applikation `ca` *unterscheidet zwischen Groß-/Klein-Schreibung in Zertifikaten und Anforderungen!* Mit anderen Worten, es ist möglich, ein inhaltlich gleiches Zertifikat zweimal herauszugeben; z.B. einmal mit Locality Name `Hamburg` und das andere Mal mit `HAMBURG`. Auch ist es möglich, daß sich vielleicht nur ein zusätzliches Leerzeichen eingeschlichen hat. Die Zertifikate würden sich jedoch durch die Seriennummer und die unterschiedliche Signatur unterscheiden.

Die Erzeugung eines Root-CA-Zertifikat ist in OpenSSL besser gelöst als in SSLeay, da es jetzt nicht mehr notwendig ist, Extensions nachträglich in das Zertifikat zu patchen. Allerdings wird bei der Erzeugung eines Root-Zertifikats kein Eintrag in die Index-Datei (`index.txt`) vorgenommen. Ebenso ist die Seriennummer des Root-Zertifikats auf „00“ festgelegt. Beides ist nicht immer wünschenswert.

Zertifikate werden von OpenSSL über Hash-Werte, die über den Distinguished Name (und evtl. die E-Mail-Adresse) gebildet werden, identifiziert. Probleme können auftreten, wenn ein Root-Zertifikat erneuert werden muß und der Distinguished Name nicht geändert werden kann oder soll. Die Konsequenzen sind offensichtlich. *Verläßt sich eine Anwendung (womit hier nicht nur OpenSSL gemeint ist) zur Identifizierung des Root-Zertifikats nur auf den Hash-Wert des Issuer-DN (aus dem zu prüfenden Zertifikat), und nicht zusätzlich auf die Extension Authority Key Identifier, kann das Root-Zertifikat nicht mehr (eindeutig) identifiziert werden.* Die Überprüfung von Zertifikatketten würde dann vermutlich fehlschlagen.

Ein Verzeichnis `newcerts` wird bei Aufruf von `make install` nicht angelegt. Die Default-Konfigurationsdatei erwartet dieses Verzeichnis aber. Das Verzeichnis muß daher nachträglich angelegt werden.

Ungünstig ist, daß der Pfad des Zertifikatverzeichnisses (z.B. `/usr/local/etc/ssl`) in die Bibliothek `libcrypto.a` einkompiliert wird. Einige Applikationen, z.B. `pkcs12` zum Lesen von Zertifikatketten, greifen auf diese Pfade zu. Das erschwert die Verwaltung von mehreren unabhängigen Zertifikatverzeichnissen oder die Änderung dieses Verzeichnisses.

Ansonsten ist die Verwaltung von Zertifikaten und der Betrieb einer CA ist mit OpenSSL gut möglich. An einigen Stellen ist allerdings Handarbeit notwendig, die aber auch teilweise als Cron-Job mit Hilfe von Skripten erledigt werden kann. Dazu zählen folgende Punkte:

- Neu herausgegebene Zertifikate müssen über ihren Hash-Wert „verlinkt“ werden.
- Es gibt keinen Automatismus, der die Indexdatei auf abgelaufene Zertifikate überprüft und Einträge entsprechend markiert.

## 8.2 Zertifikate und Browser

Durch die OpenSSL-Applikation `pkcs12` können Anwendungs- und CA-Zertifikate in die Browser importiert werden. Das Programm arbeitet nur mit dem Navigator ab Vers. 4.04 und dem Explorer ab Vers. 4.0 zusammen. Für ältere Browser-Versionen muß das PFX-Programm siehe Anhang (9.1) verwendet werden. Außerdem ist es möglich, Benutzerzertifikate mit Schlüssellängen bis zu 2048 Bit als `.p12`-Datei in eine Export-Version des Navigators 4.x zu laden. *Da die USA die Export-Beschränken für Kryptosoftware gelockert haben, sollten demnächst Browser-Versionen zur Verfügung stehen, die diese Beschränkung der Schlüssellänge nicht mehr haben.*

PKCS-12 ist ein Standard, der ein Format beschreibt in dem Zertifikate und Schlüssel außerhalb von Anwendungen gespeichert werden können. Zertifikate und Schlüssel, die in diesem Formate vorliegen, können dann von verschiedenen Anwendungen gelesen bzw. in diesem Format geschrieben (exportiert) werden.

Neben dem Import von Zertifikaten besteht auch die Möglichkeit, das aus den Browser im PKCS-12-Format exportierte Zertifikate für OpenSSL verfügbar zu machen.

### 8.2.1 Export aus dem Browser

Um beispielsweise ein mittels der Netscape Export-Funktion aus dem Browser exportiertes Zertifikat *cert*`.p12` für OpenSSL zugänglich zu machen, ist folgender Befehl notwendig:

```
pkcs12 -in cert.p12 -out cert.pem [-des3 | -des | -idea]
```

Es wird nach einem Import-Passwort gefragt; es ist das Passwort, das beim Zertifikat-Export mit Netscape angegeben werden mußte und mit dem *cert*`.p12` verschlüsselt wurde. Die Optionen `-des3`, `-des`, bzw. `-idea` geben an, ob und wie das Zertifikat und der Schlüssel, welche sich nach Ausführung des obigen Kommandos in *cert*`.p12` befinden, verschlüsselt werden sollen. Es wird dazu wieder nach einem Passwort gefragt.

### 8.2.2 Import in den Browser

Um ein Zertifikat in ein vom Browser importierbares Format zu wandeln:

```
pkcs12 -export -name Listbox-Name -in cert.pem -inkey key.pem -out file.p12
```

*Listbox-Name* ist der Bezeichner, unter dem das importierte Zertifikat später in einer der Security-Rubriken des Browsers geführt werden soll. Nach dem die Datei *file*`.p12` erzeugt wurde, kann sie über die Import-Funktion der Browser importiert werden.

Zusammen mit einem Benutzer-Zertifikat können auch CA-Zertifikate in einen Browser importiert werden. Dazu werden Benutzer-Zertifikat und die CA-Zertifikate durch das `pkcs12`-Kommando in eine Datei geschrieben. Das folgende Kommando funktioniert allerdings nur, wenn das Verzeichnis `ssl/certs` unterhalb des Pfades, der bei der Konfiguration des Paketes (siehe oben (3.3)) angegeben wurde, vorliegt. Der Pfad ist in

die Bibliothek `libcrypto.a` einkompiliert und kann nicht geändert werden. Es müssen alle CA-Zertifikate der Kette (auch das Root-Zertifikat) im Verzeichnis `ssl/certs` vorliegen und über ihre Hash-Werte (siehe oben (6.3)) verlinkt sein.

Um eine Zertifikat-Kette in ein vom Browser importierbares Format zu wandeln:

```
pkcs12 -chain -export -name Listbox-Name -in cert.pem -inkey key.pem -out file.p12
```

Sollen die CAs der Zertifikatkette ebenfalls eine Bezeichnung bekommen kann dazu die Option `-caname "`*CA Name*`"`, auch mehrfach, verwendet werden. Damit die Zuordnung der Bezeichner zu den CAs stimmt, muß als erstes der Bezeichner für die Root-CA und die nächsten Bezeichner entsprechend der Kette aufgeführt werden. Dieser sogenannte „Friendly Name“ für CA-Zertifikate wird von Microsoft-Produkten unterstützt.

Statt mit der Option `-chain` kann ein CA-Zertifikat über die Option `-certfile` *cacerts.pem* angegeben werden. Sollen es mehrere sein, werden sie mit in die Datei geschrieben, z.B. durch folgendes Kommando:

```
cat ca1.pem ca2.pem ca3.pem > cacerts.pem
```

Nachdem die `.p12`-Datei erzeugt wurde, kann das Benutzer-Zertifikat in den Browser importiert werden und damit gleichzeitig die CA-Zertifikate. Das Benutzer-Zertifikat kommt in die Rubrik „Security/Yours“ und die CA-Zertifikate nach „Security/Signers“. Für die CA-Zertifikate muß dann mittels des „Edit“-Buttons im NSC noch mitgeteilt werden, wofür das CA-Zertifikat gültig sein soll. Es gibt drei Möglichkeiten zur Auswahl: Zertifizierung von SSL-Servern, E-Mail-Benutzern oder Software-Entwicklern. Diese Auswahl gilt auch für Wurzel-Zertifikate, die nur untergeordnete CAs signieren, wobei dann die Auswahl inhaltlich wenig Sinn macht. Wenn das Wurzel-Zertifikat für einen Verwendungszweck als gültig akzeptiert wurde muß eine Auswahl für untergeordnete CAs nicht mehr getroffen werden, auch wenn es hier u.U. Sinn machen würde. Erst nach Auswahl des Verwendungszwecks verläuft eine Überprüfung des Benutzer-Zertifikats erfolgreich.

Auf diese Weise können aber nur CA-Zertifikate mit gesetzten „CA-Bit“ importiert werden, also wenn `nsCertType` auf `sslCA`, `emailCA`, `objCA`, `basicConstraints` bzw. deren Kombinationen im CA-Zertifikat gesetzt sind.

## 8.3 Zertifikate und CRLs mit dem Netscape Browser

Zertifikate und CRLs können von einem HTTP-Server als spezielle MIME-Types an einen Browser gesendet werden. Dazu müssen die Zertifikate und die CRLs im (binären) DER-Format vorliegen. Üblicherweise erzeugen die OpenSSL-Applikationen Dateien im PEM-Format (Base64). Es besteht aber die Möglichkeit, die Dateien nachträglich in das DER-Format umzuwandeln. Zertifikate werden durch folgenden Befehl in das DER-Format umgewandelt:

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

Der entsprechende Befehl, um eine CRL in das DER-Format zu wandeln, lautet:

```
openssl crl -in crl.pem -outform der -out crl.der
```

*Achtung:*
Netscape-Browser akzeptieren (bisher) keine X.509v2-CRLs. Genaueres steht im Abschnitt über CRLs (7).

### 8.3.1 Zertifikate

Mit OpenSSL erzeugte Zertifikate lassen sich in den Netscape Communicator/Navigator (NSC) relativ einfach importieren. Das Zertifikat im (binären) DER-Format kann von einem HTTP-Server als einer der folgenden MIME-Types an den Browser geschickt werden:

- `application/x-x509-email-cert`
- `application/x-x509-user-cert`
- `application/x-x509-ca-cert`

Dazu werden die Zertifikate auf einen Server zum Herunterladen zur Verfügung gestellt. Die Datei `mime.types` des jeweiligen Web-Servers (z.B. Apache) muß um die obigen Typen ergänzt werden. Die Dateiendung, die den MIME-Types zugeordnet wird, muß mit der Dateiendung der Zertifikate übereinstimmen. Der Server kennzeichnet dann beim Senden die Datei mit dem entsprechenden MIME-Type. Die MIME-Types signalisieren dem Browser, daß es sich um ein Zertifikat handelt, und der Browser startet dann einen entsprechenden Interaktions-Modus. Der Browser bietet dem Benutzer in Abhängigkeit vom MIME-Type und der Erweiterung *Netscape-Cert-Type* eine Auswahl an, für welchen Zweck ein Zertifikat bestimmt ist. Im folgenden eine Aufstellung der *Browser-Vorschläge* `<http://home.netscape.com/eng/security/comm4-cert-download.html>` in Abhängigkeit von Cert- und MIME-Type.

**Zertifikate als MIME-Type `x-x509-ca-cert` gesendet:**

| `nsCertType` | Beschreibung | Rubrik | Accept this CA for Certifying ... |
|---|---|---|---|
| `objCA` | CA-Zert. für 0x10-Zert. | Signers | ... Software-Developers |
| `emailCA` | S/MIME-CA | Signers | ... e-mail users |
| `sslCA` | SSL CA | Signers | ... network sites |
| `reserved` | Reserviert | Signers | Für alle fünf CertTypes |
| `objsign` | Zur Objekt-Signierung | Signers | folgende Auswahl: |
| `email` | S/MIME Benutzer-Zert. | Signers | ... Software-Developers |
| `server` | SSL-Server | Signers | ... network sites |
| `client` | SSL-Client | Signers | ... e-mail users |

**Zertifikate als MIME-Type `x-x509-user-cert` gesendet:**

Alle Zertifikate werden *unabhängig vom Netscape-Cert-Type* als „eigene E-Mail Benutzer-Zertifikate“ erkannt.

**Zertifikate als MIME-Type `x-x509-email-cert` gesendet:**

Alle Zertifikate werden *unabhängig vom Netscape-Cert-Type* als „fremdes E-Mail Benutzer-Zertifikat“ erkannt. Nachdem die Zertifikate akzeptiert wurden, werden sie allerdings in unterschiedliche NSC-Zertifikat-Rubriken, „People“ und „Certificate/Signers“ einsortiert. Die drei Zertifikat-Typen (`nsCertType objCA, objsign, server`), die in der Aufstellung fehlen, sind nach Akzeptieren des Zertifikats in keiner Rubrik des Browsers mehr aufzufinden.

| `nsCertType` | Beschreibung | Rubrik | Accept this CA for Certifying ... |
|---|---|---|---|
| `email` | S/MIME-CA | Signers | ... e-mail users |
| `sslCA` | SSL-CA | Signers | ... network sites |
| `sslCA` | SSL-CA | | ... e-mail users |
| `reserved` | Reserviert | People | |
| `email` | S/MIME Benutzer-Zert. | People | |
| `client` | SSL-Client | People | |

Der NSC akzeptiert Schlüssellängen von 2048 Bit für CA-Zertifikate. Für Benutzerzertifikate beträgt die max. Schlüssellänge 512 Bit bei der Export-Version des Browsers. Im allgemeinen werden CA-Zertifikate von einem Server (die entsprechende Konfiguration des Servers vorausgesetzt) im binären DER-Format zur

Verfügung gestellt. Der Benutzer kann dann das CA-Zertifikat als MIME-Type `x-x509-ca-cert` (in der Regel über eine HTML-Seite) in den Browser laden. Er wird dabei durch einen Interaktionmodus geführt, in dessen Verlauf er bestimmen kann, welches Vertrauen er dem Zertifikat entgegen bringt. Laut Stephen N. Henson `<http://www.drh-consultancy.demon.co.uk/caornot.html>` *kann jedes Zertifikat, unabhängig von der Art der enthaltenen Extensions, nach Durchlaufen des Interaktions-Modus als CA für jeden Zweck akzeptiert werden.* Das kann dann zu Problemen führen, wenn die CA für einen Zweck akzeptiert wurde, den die Extensions im CA-Zertifikat nicht unterstützen. Ist in dem CA-Zertifikat z.B. kein Bit gesetzt, welches die CA als Herausgeber von S/MIME-Zertifikaten kennzeichnet, und wird ein von dieser CA herausgegebenes Zertifikat zum Signieren von E-Mail verwendet, wird die Mail beim Empfänger wegen des ungültigen CA-Zertifikats abgewiesen.

Die Möglichkeit, CA-Zertifikate über Benutzerzertifikate ohne Web-Server in den NSC zu importieren, wird im Abschnitt über PFX (9.1) bzw. PKCS12 (8.2) beschrieben.

Eine Schlüsselerzeugung mit anschließender Online-Zertifizierung wird unterstützt. Da der Schlüssel vom Browser erzeugt wird, ist hier bei der Export-Version die Schlüssellänge auf 512 Bit beschränkt. Bei Verwendung der oben genannten Programme ist es möglich, die max. Schlüssellänge für Benutzerzertifikate auf 2048 Bit zu erhöhen (getestet mit Version 4.0 und 4.05). SSL-Server-Zertifikate werden beim Anwählen eines SSL-Servers automatisch geladen. Liegt das CA-Zertifikat für einen solchen Server nicht vor, wird ebenfalls ein Interaktionsmodus gestartet, in dessen Verlauf der Benutzer selber entscheiden muß, welches Vertrauen er dem Server-Zertifikat entgegenbringt. Liegt dagegen das CA-Zertifikat vor, bekommt der Benutzer in Abhängigkeit von der vorgenommenen Einstellung am Browser keine oder nur eine kleine Meldung. (Es sei an dieser Stelle darauf hingewiesen, daß der Server beim Verbindungsaufbau die Möglichkeit hat, zusätzlich zu seinem eigenen Zertifikat auch die Zertifikate der übergeordneten CA(s) mitzuliefern.)

### 8.3.2 CRLs

*Achtung:*
Netscape-Browser akzeptieren (bisher) keine X.509v2 CRLs. Genaueres steht im Abschnitt über CRLs (7).

Leider scheint der Netscape-Browser die Extension CRL Distribution Points nicht zu unterstützen. Eine CRL muß daher explizit vom Anwender geladen werden. Eine CRL kann nur in den Browser geladen werden, wenn die CRL durch den Anwender von einem Server als spezieller MIME-Type runtergeladen wird. Diese Mechanismus wird im folgenden beschrieben. Die aufgeführten MIME-Types müssen wieder, wie im vorigen Abschnitt (8.3.1) beschrieben, eigentragen sein. Ebenso müssen die Dateiendungen übereinstimmen.

Die im folgenden verwendeten CRLs waren vom Typ X.509v1 und lagen im DER-Format vor.

**CRL als MIME-Type `application/x-pkcs7-crl` gesendet:**

Der Browser (Vers. 4.03) akzeptiert die CRL, ohne eine Meldung auszugeben. Wird versucht, die CRL ein zweites Mal zu laden, wird die folgende Error-Box angezeigt:

Nachdem in einem Browser Vers. 4.05 eine CRL geladen wurde, taucht im Security Fenster nach Anwahl der Rubrik „Signers“ ein neuer Button „Edit/View CRL's “ auf. Darüber lassen sich CRLs anzeigen, löschen und neu laden.

**CRL als MIME-Type `application/x-x509-crl` gesendet:**

Der Browser (Vers. 4.03) lädt das CRL-Binary nicht als CRL, sondern als Text mit entsprechend kryptischen Zeichen im Browser-Fenster.

Wurde eine CRL, die ein widerrufenes SSL-Server Zertifikat enthielt, in einen Browser der Vers. 4.05 gebracht, wird anschließend eine Verbindung zu dem entsprechenden SSL-Server verweigert. Allerdings mit einer wenig hilfreichen Meldung der Art „Es gibt Schwierigkeiten...“

Der Browser gibt folgende Meldung aus, wenn der Gültigkeitszeitraum einer schon geladenen CRL überschritten ist:


Es muß dann zunächst die aktuelle CRL geladen werden, bevor erneut eine Verbindung aufgebaut werden kann.

## 8.4 Zertifikate und CRLs mit Microsoft Browser

Der Internet Explorer (MSIE) unterstützt u.a. die Standard X509v3-Attribute wie Basic Constraints und Key Usage (siehe jedoch Zertifizieren (6.3)). Um ein CA-Zertifikat für den MSIE als solches zu kennzeichnen, muß das „CA-Bit“ in Form von Basic Constraints gesetzt werden. Außerdem kann über Key Usage die Verwendung des Zertifikats eingeschränkt werden. Der Browser interpretiert laut MS-Dokumentation Key Usage immer, egal ob Critical oder nicht.

Die von Steve P. Henson geschilderten Probleme mit Critical Extensions, scheinen zumindest für MS Outlook Express 5 nicht mehr zu gelten. Eine Zertifikat-Kette mit Critical gesetzten Basic Constraints ließ sich problemlos importieren.

Die folgende Angaben beziehen sich hauptsächlich auf den MSIE 4.0, und zum Teil auf Version 5.x. Es ist dabei zu berücksichtigen, daß sich das Verhalten der Browsers unter Windows NT in Abhängigkeit vom installierten Servicepack ändern kann. Gleiches gilt vermutlich auch für Windows 9x.

### 8.4.1 Zertifikate

Root-CA-Zertifikate können über einen Web-Server als MIME-Type `application/x-x509-ca-cert` gesendet werden (s.o. (8.3.1)). Sie werden als CA-Zertifikate für „Netzwerk-Client-“, „Netzwerk-Server-Authentifikation“, „Sichere-E-Mail“ und „Software-Herausgeber“ akzeptiert. Es besteht die Möglichkeit, über Kontrollkästchen die Beschränkungen des Zertifikats einzeln zu (de-)aktivieren. Nach Akzeptieren findet sich das Zertifikat in der Rubrik „Ansicht / Internetoptionen / Inhalt / Agenturen“.

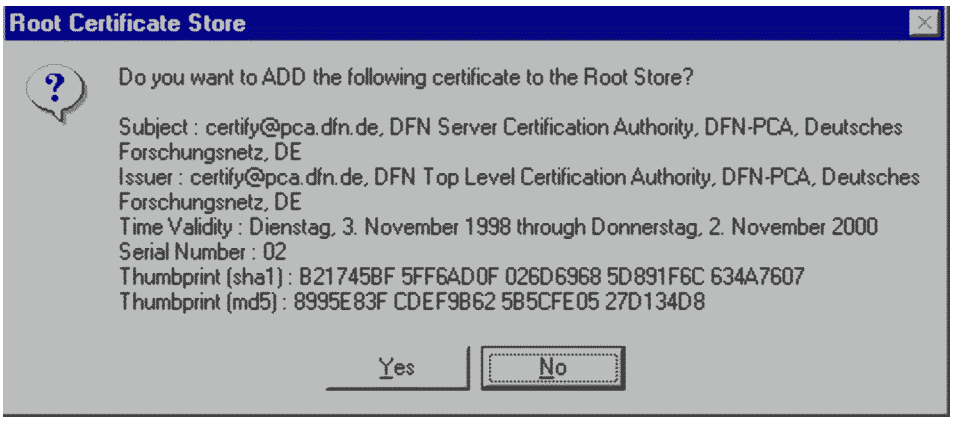
Erhebliche Probleme gibt es allerdings mit CA-Zertifikaten, die nicht selbstsigniert, also keine Root-Zertifikate sind. Bis Version 4.x läßt sich ein solches Zertifikat nur durch eine der folgenden Improvisationen installieren:

1. Das Zertifikat wird als MIME-Type `application/pkcs7-mime` auf einen Server zur Verfügung gestellt. Zur Installation des MIME-Types siehe obigen Abschnitt (8.3.1). Bei dem runterladen des Zertifikats wird das MS Address Book `wab.exe` aufgerufen, wenn es installiert ist. Danach besteht die Möglichkeit das Zertifikat über eine Dialogbox als gültig zu akzeptieren. Das Zertifikat erscheint dann zwar nicht in der Liste der vertrauenswürdigen CAs, aber von dieser CA herausgegebene Server-Zertifikate werden als gültig anerkannt.

2. Soll das CA-Zertifikat in der Liste der vertrauenswürdigen CAs erscheinen, wird das MS-Programm `certmgr.exe` benötigt. Das CA-Zertifikat muß von einem Server als MIME-Type `application/x-x509-ca-cert` heruntergeladen und lokal abgespeichert werden, z.B. unter dem Namen `cacert.crt`. Jetzt mußeine DOS-Shell gestartet werden. In der DOS-Shell wird in das Verzeichnis gewechselt in dem `certmgr.exe` und `cacert.crt` vorliegen. Jetzt wird das Zertifikat durch folgendes Kommando geladen (*Reihenfolge der Parameter unbedingt beachten*):

   ```
   certmgr -c cacert.crt -s Root -add
   ```

   Hat das Importieren des Zertifikats fehlerfrei funktioniert, wird die Meldung „CertMgr Succeeded“ ausgegeben und es erscheint folgende Dialogbox, die durch das Klicken auf „Yes“ zu beenden ist:




Mit dem IE Version 5.x ist dieses Verfahren nicht mehr notwendig. Es gibt jetzt eine Rubrik für nicht selbstsignierte CAs. Die Version 5.x zeichnet sich gegenüber der 4er-Version durch eine grundsätzlich verbesserte Verwaltung von Zertifikaten aus. Allerdings werden vom Benutzer akzeptierte *Server*-Zertifikate, in die Rubrik „Eigene Zertifikate“ einsortiert, es gibt keine Rubrik für Server-Zertifikate.

Benutzer-Zertifikate können auf zwei Arten in den MSIE gebracht werden. Die eine Art umfaßt die Schlüssel- und Request-Erzeugung mit den Browser durch Ausführen eines Java- oder VB-Skript. Anschließend wird der Request online zertifiziert. Auf diese Methode wird hier nicht weiter eingegangen.

Die zweite Möglichkeit besteht darin, wie im Abschnitt Erzeugen von Requests (6.2) beschrieben, einen mit OpenSSL erzeugten Request zu signieren und diesen anschließend als `.p12`-Datei in den Browser zu importieren. Dazu wird in der Rubrik „Ansicht / Internetoptionen / Inhalt / Eigene“ eine Dialogbox geöffnet, wo über den Button „Importieren“ ein Zertifikat importiert werden kann. Die in den Browser gebrachten Zertifikate stehen im MS-Mail Programm Outlook-Express (OE) zur Verfügung, wenn die im Zertifikat angegebene E-Mail-Adresse mit der im OE übereinstimmt.

Beim Versuch eine `.p12`-Datei in die 5er-Version des Browsers bzw. Outlook Express zu laden, fällt auf, daß diese die Endung `.p12` nicht kennen. Erkannt wird die Endung `.pfx`. Die `.p12`-Datei läßt sich aber trotzdem laden, wenn die Ansicht auf „alle Dateien“ erweitert, und die Datei dann ausgewählt wird. Eine `.p12`-Datei läßt sich, zumindest unter Windows 98 auch durch einen „Doppelklick“ auf die Datei laden. Es wird dann der „Internet-Assistent für die Zertifikatverwaltung“ gestartet.

Unglücklicherweise werden persönliche Zertifikate und Private-Keys durch den 4er IE *nicht geschützt*. Der einzige Schutz besteht in der Zugangsbeschränkung durch das Betriebssystem. Stephen N. Henson beschreibt in einem *Dokument* `<http://www.drh-consultancy.demon.co.uk/cexport.html>` wie der Schutz des Private-Keys durch den MS `certmgr` nach dem Installieren eines Zertifikats erhöht werden kann.

### 8.4.2 CRLs

CRLs bezieht der MSIE über eine im Zertifikat angegebene URI. Die URI wird über die Extension CRL Distribution Points in ein Zertifikat gebracht. Eine Zertifikatprüfung erfolgt erst, wenn im 4er-Browser in der Rubrik „Ansicht / Internetoptionen / Erweitert“ im Abschnitt „Sicherheit“, das Kontrollkästchen „Auf zurückgezogene Zertifikate überprüfen“ angewählt ist.

In der 5er-Version heißt die Rubrik „Extras / Internetoptionen / Erweitert“. Dort sind die Kontrollkästchen „Auf zurückgerufenen Serverzertifikate“ bzw. „Zertifikate von Herausgebern prüfen“ anzuwählen. Damit OE 5 auf zurückgerufene Benutzer-Zertifikate prüft, muß in der Rubrik „Extras / Optionen / Sicherheit / Weitere Einstellungen“ das Kästchen „Auf zurückgezogene Digitale ID's prüfen“ angewählt werden. Die CRLs werden dann automatisch über die entsprechende URL bezogen.

Es ist auch möglich, CRLs über die Import-Funktion des Zertifikat-Assistenten (OE 5) zu Laden. Der Assistent erkennt an dem Datei-Inhalt, daß es sich um eine CRL handelt. Leider konnte nirgendwo eine Möglichkeit entdeckt werden, die geladenen CRLs anzeigen zu lassen. Auch kann der Inhalt der CRLs nicht angezeigt werden.

Die Handhabung von CRLs scheint auch noch nicht ganz ausgereift zu sein. Ein Benutzer-Zertifikat wurde vom Zertifikat-Assistenten als nicht überprüfbar markiert, da keine CRL der CA vorlag. Die CA war eine Zwischen-CA *für die aber eine CRL geladen war*. Für die Root-CA, die die Zwischen-CA zertifiziert hat, lag dagegen keine CRL vor...

# 9 Ergänzende Programme

## 9.1 `pfx-0.1.2` von Stephen N. Henson

Das Programm ermöglicht es, vom Netscape Communicator/Navigator (NSC) und MS-Internet-Explorer (MSIE) (jeweils ab Vers. 4.0x) exportierte Zertifikate für OpenSSL lesbar zu machen, bzw. mit OpenSSL erzeugte Zertifikate in den NSC und den MSIE zu importieren. Außerdem ist es möglich, Benutzerzertifikate mit Schlüssellängen bis zu 2048 Bit als `.p12`-Datei in den NSC zu laden. Für NSC ab Vers. 4.04 und MSIE ab Vers. 4.0x sollte statt `pfx` das `pkcs12`-Programm (siehe PKCS12 (8.2)) verwendet werden.

### 9.1.1 Übersetzung und Installation

Folgende Änderungen sind im `Makefile` vorzunehmen:

`($(SSLDIR)=`*`Installationsverzeichnis`*`, $(SSLSRC)=`*`Quellverzeichnis`*`)`

- Zeile 1, `SSLINC=/usr/local/ssl/include`:
  Wenn die Include-Dateien von OpenSSL mit installiert wurden, Pfad auf `$(SSLDIR)/include`, sonst Pfad auf `$(SSLSRC)/include` setzen.
- Zeile 2, `SSLLIB=/usr/local/ssl/lib`:
  Pfad auf `SSLLIB=$(SSLDIR)/lib` setzen.

- Zeile 3, `CFLAGS=-Wall -O2 -I$(SSLINC) -g`:
  Flags ändern in

  ```
  CFLAGS=-fomit-frame-pointer -mv8 -Wall -O2 -I$(SSLINC)
  ```

- In Zeile 11, `cc -g -L$(SSLLIB) -o pfx $(OBJS) -lcrypto`:
  `cc` durch `gcc` ersetzen und die Option `-g` entfernen.

Schließlich ist das Programm mit

```
make 'CC=gcc'
```

zu übersetzen.

Als Installationsverzeichnis bietet sich `$(SSLDIR)/bin` an:

- `cp $(PFX_SRC)/pfx $(SSLDIR)/bin`
- `chmod 755 $(SSLDIR)/bin/pfx`

### 9.1.2 Anwendung von `pfx`

**Export aus einen Browser**

Um ein mittels der Netscape Export-Funktion aus dem Browser exportiertes Benutzer-Zertifikat *cert.p12* für OpenSSL zugänglich zu machen, ist folgender Befehl notwendig:

```
pfx -in cert.p12 -out cert.pem [-des3 | -des | -idea]
```

Es wird nach einem Import-Passwort gefragt; es ist das Passwort, das beim Zertifikat-Export mit Netscape angegeben werden mußte und mit dem *cert.p12* verschlüsselt wurde. Die Optionen `-des3`, `-des`, bzw. `-idea` geben an, ob und wie das exportierte Zertifikat verschlüsselt werden soll.

**Import in einen Browser**

Um ein mit OpenSSL erstelltes Zertifikat in den NSC zu importieren:

```
pfx -export -name Listbox-Name -out cert.p12 -in cert.pem
```

*Listbox-Name* ist die Bezeichnung, unter der das Zertifikat nach dem Import im Browser in einer Security-Rubrik geführt wird.

Nach dem Schlüssel-Passwort wird ein weiteres Passwort erfragt; es ist das Passwort um die *cert.p12*-Datei zu verschlüsseln. Das so vorbereitete Zertifikat kann über NSC Import-Funktion importiert werden. Der Browser erwartet als zweites Passwort (das erste diente zum Freigeben der NSC Zertifikat-Datenbank) das Passwort, mit dem die `.p12`-Datei verschlüsselt wurde. Die Zertifikate werden nur in den NSC-Bereich „Security/Yours“ also als „eigene E-Mail-Zertifikate“, gebracht.

Mit folgendem Befehl kann indirekt auch ein CA-Zertifikat in den Netscape-Browser geladen werden:

```
pfx -export -name Listbox-Name -in usercert.pem -inkey userkey.pem -out mycert.p12
-certfile CAcert.pem
```

Das CA-Zertifikat wird an ein vorher erzeugtes Benutzer-Zertifikat *usercert.pem* gebunden. Anschließend kann das Benutzer-Zertifikat importiert werden und damit gleichzeitig das „angehängte“ CA-Zertifikat. Das Benutzer-Zertifikat kommt in die Rubrik „Security/Yours“ und das CA-Zertifikat nach „Security/Signers“. Für das CA-Zertifikat muß dann mittels des „Edit“-Buttons im NSC noch mitgeteilt werden, wofür das CA-Zertifikat gültig sein soll. Erst dann verläuft eine Überprüfung des Benutzer-Zertifikats erfolgreich. Auf diese Weise können aber nur Zertifikate mit gesetzten „CA-Bit“ importiert werden, also wenn `nsCertType` auf `sslCA`, `emailCA`, `objCA` bzw. deren Kombinationen gesetzt ist.

Weiter ist es möglich, Zertifikat-Ketten in den Browser zu importieren. Voraussetzung ist, daß alle n-1 Zertifikate das „CA-Bit“ gesetzt haben. Mit folgenden Befehl werden an ein Zertifikat alle in der Zertifikat-Hierarchie darüber liegenden Zertifikate, einschließlich dem Root-CA-Zertifikat, an das Zertifikat gebunden:

```
pfx -export -name Listbox-Name -in usercert.pem -inkey userkey.pem -out usercert.p12
-chain
```

Das Kommando funktioniert nur erfolgreich, wenn die CA-Zertifikate nach `$(SSLDIR)/certs` kopiert und deren Hash-Werte gebildet worden sind (s. die *Anmerkung* am Ende des Abschnitts über die Hash-Werte (6.1)). Die mit einem Zertifikat verbundenen CA-Zertifikate werden alle in den Browser gebracht und sind über die Browser-Rubrik „Signers“ zugänglich. Eine erfolgreiche Überprüfung der Zertifikate setzt voraus, daß zumindest für das Root-CA-Zertifikat der „CA-Typ“ (Server-CA, Client-CA usw.) mit Hilfe des Browsers eingestellt wird (s.o., Edit-Button (9.1.2)).

Eine Liste aller Optionen findet sich im Anhang PFX (9.1).

### 9.2 `pkcs12-054` von Stephen N. Henson

Das Programm `pkcs12-054` bzw. dessen bereitgestellte Funktionalität ist in OpenSSL-0.9.5-dev vollständig integriert (siehe oben (8.2)). Somit wird `pkcs12-054` nicht mehr benötigt.

### 9.3 `ca-fix-0.3` von Stephen N. Henson

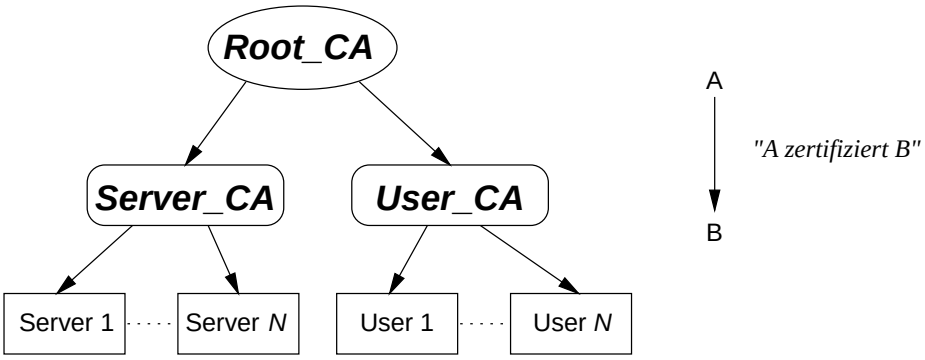
Die von dem Programm CA-Fix bereitgestellte Funktionalität ist in OpenSSL-0.9.5-dev vollständig integriert. Somit wird CA-Fix nicht mehr benötigt.

## A openssl.cnf - Beispiel-Datei

Die folgende Beispiel-Konfigurationsdatei enthält drei Abschnitte für verschiedene CA-Konfigurationen. Der erste Abschnitt `[ Root_CA ]` enthält eine Konfiguration zur Herausgabe von CA-Zertifikaten, entsprechend `[ Server_CA ]` zur Herausgabe von SSL-Server-Zertifikaten (*Server 1 bis N*) und `[ User_CA ]` für die Herausgabe von Benutzer-Zertifikaten (*User 1 bis N*). Damit kann die unten dargestellte Zertifizierungshierarchie realisiert werden.

Die einzelnen CA-Abschnitte unterscheiden sich vor allem in der Angabe zum Extension-Abschnitt, der beim Schlüsselwort `x509_extensions` im jeweiligen CA-Abschnitt festgelegt ist. Über den Extension-Abschnitt wird bestimmt, welche Extensions die herausgegebenen Zertifikate enthalten und somit ihr Verwendungszweck. Die CA-Abschnitte werden bei der der Zertifizierung von Requests durch das `ca`-Kommando mit der Option `-name` angewählt (siehe Abschnitt Signieren (6.3)).

Für die drei CA-Abschnitte gemeinsame Werte können auch am Anfang der Konfigurationsdatei vor dem ersten Abschnitt (hier `[ new_oids ]` festgelegt werden.




```
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#

# RANDFILE              = $ENV::HOME/.rnd
# oid_file              = $ENV::HOME/.oid
# oid_section           = new_oids

# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions            =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)

pfad                    = /usr/local/etc/ssl

[ new_oids ]

# We can add new OIDs in here for use by 'ca' and 'req'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6

####################################################################

[ ca ]

default_ca      = Server_CA             # The default ca section

####################################################################


[ Root_CA ]         # Abschnitt fuer eine Root CA

dir             = $pfad/PCA               # Where everything is kept
certs           = $dir/certs              # Where the issued certs are kept
```

```
crl_dir         = $dir/crl               # Where the issued crl are kept
database        = $dir/index.txt         # database index file.
new_certs_dir   = $dir/newcerts          # default place for new certs.

certificate     = $dir/PCAcert.pem       # The CA certificate
serial          = $dir/serial            # The current serial number
crl             = $dir/crl.pem           # The current CRL
private_key     = $dir/private/PCAkey.pem # The private key
RANDFILE        = $dir/private/.rand     # private random number file

x509_extensions = PCA_ext                # The extentions to add to the cert

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
#crl_extensions = crl_ext                # Extensions to add to CRL

default_days    = 730                    # how long to certify for
default_crl_days= 30                     # how long before next CRL
default_md      = md5                    # which md to use.
preserve        = no                     # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy          = policy_match

#####################################################################

[ Server_CA ]         # Abschnitt fuer eine Server CA

dir             = $pfad/SCA              # Where everything is kept
certs           = $dir/certs             # Where the issued certs are kept
crl_dir         = $dir/crl               # Where the issued crl are kept
database        = $dir/index.txt         # database index file.
new_certs_dir   = $dir/newcerts          # default place for new certs.

certificate     = $dir/SCAcert.pem       # The CA certificate
serial          = $dir/serial            # The current serial number
crl             = $dir/crl.pem           # The current CRL
private_key     = $dir/private/SCAkey.pem # The private key
RANDFILE        = $dir/private/.rand     # private random number file

x509_extensions = SCA_ext                # The extentions to add to the cert
#crl_extensions = crl_ext                # Extensions to add to CRL
default_days    = 365                    # how long to certify for
default_crl_days= 30                     # how long before next CRL
default_md      = md5                    # which md to use.
preserve        = no                     # keep passed DN ordering

policy          = policy_anything
```

```
#####################################################################

[ User_CA ]         # Abschnitt fuer eine User CA

dir             = $pfad/UCA               # Where everything is kept
certs           = $dir/certs              # Where the issued certs are kept
crl_dir         = $dir/crl                # Where the issued crl are kept
database        = $dir/index.txt          # database index file.
new_certs_dir   = $dir/newcerts           # default place for new certs.

certificate     = $dir/UCAcert.pem        # The CA certificate
serial          = $dir/serial             # The current serial number
crl             = $dir/crl.pem            # The current CRL
private_key     = $dir/private/UCAkey.pem # The private key
RANDFILE        = $dir/private/.rand      # private random number file

x509_extensions = UCA_ext                 # The extentions to add to the cert
#crl_extensions = crl_ext                 # Extensions to add to CRL
default_days    = 365                     # how long to certify for
default_crl_days= 30                      # how long before next CRL
default_md      = md5                     # which md to use.
preserve        = no                      # keep passed DN ordering

policy          = policy_anything

#####################################################################


# For the CA policy
# Auch hier gilt:
# ... you must list all acceptable 'object' types.

[ policy_match ]

countryName             = match
stateOrProvinceName     = supplied
localityName            = optional
organizationName        = supplied
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.

[ policy_anything ]

countryName             = match
```

```
stateOrProvinceName    = optional
localityName           = optional
organizationName       = optional
organizationalUnitName = optional
commonName             = supplied
emailAddress           = optional

####################################################################

[ req ]

default_bits        = 1024
default_keyfile     = privkey.pem
distinguished_name  = req_distinguished_name
attributes          = req_attributes
x509_extensions     = v3_ca     # The extentions to add to the self signed cert

# Passwords for private keys if not present they will be prompted for
# input_password = secret
# output_password = secret

# This sets a mask for permitted string types. There are several options.
# default: PrintableString, T61String, BMPString.
# pkix   : PrintableString, BMPString.
# utf8only: only UTF8Strings.
# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
# MASK:XXXX a literal mask value.
# WARNING: current versions of Netscape crash on BMPStrings or UTF8Strings
# so use this option with caution!
string_mask = nombstr

# req_extensions = v3_req # The extensions to add to a certificate request

####################################################################

[ req_distinguished_name ]

countryName                     = Country Name (2 letter code)
countryName_default             = DE
countryName_min                 = 2
countryName_max                 = 2

stateOrProvinceName             = State or Province Name (full name)
#stateOrProvinceName_default    = Schleswig-Holstein

localityName                    = Locality Name (eg, city)
#localityName_default           = Kiel

0.organizationName              = Organization Name (eg, company)
#0.organizationName_default     = Universitaet Kiel
```

```
# we can do this but it is not needed normally :-)
#1.organizationName             = Second Organization Name (eg, company)
#1.organizationName_default     = World Wide Web Pty Ltd

organizationalUnitName          = Organizational Unit Name (eg, section)
#organizationalUnitName_default = Studis

commonName                      = Common Name (eg, YOUR name)
commonName_max                  = 64

emailAddress                    = Email Address
emailAddress_max                = 60

# SET-ex3                       = SET extension number 3

####################################################################

[ req_attributes ]

# Das Challenge Password dient dazu, sich bei Verlust des geheimen Schluessels
# gegenueber der Herausgeber-CA fuer einen Zertifikatwiderruf auszuweisen.
# Wird bei Erstellung der Zertifikat-Anforderung erfragt.

challengePassword               = A challenge password
challengePassword_min           = 4
challengePassword_max           = 20

unstructuredName                = An optional company name

####################################################################

[ PCA_ext ]

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.
basicConstraints         = critical, CA:TRUE

# Moeglich: digitalSignature, nonRepudiation, keyEncipherment,
#           dataEncipherment, keyAgreement, keyCertSign,
#           cRLSign, encipherOnly, decipherOnly
keyUsage                 = cRLSign, keyCertSign

# PKIX recommendations
subjectKeyIdentifier     = hash
authorityKeyIdentifier   = keyid,issuer:always

# Import the email address.
subjectAltName           = email:copy
```

```
# Copy subject details
issuerAltName           = issuer:copy

crlDistributionPoints   = URI:http://mystic.pca.dfn.de/PCA.crl

# Moeglich: client, server, email, objsign, reserved, sslCA, emailCA, objCA
nsCertType              = sslCA, emailCA, objCA

# Hier kann der den folgenden Url's gemeinsame Url-Stamm angegeben werden.
nsBaseUrl               = https://mystic.pca.dfn.de/

# Die Seite mit der CA-Policy
nsCaPolicyUrl           = http://www.pca.dfn.de/dfnpca/policy/wwwpolicy.html

# Ein Text, der von Netscape-Browsern angezeigt wird
nsComment               = This certificate was issued by a PCA\n\
just for testing.

# Hier kann eine Online-Zertifikatspruefung stattfinden, indem auf die
# Url in der Form ../foo.cgi?aaaa zugegriffen wird. "aaaa" ist dabei
# die ASCII-kodierte Seriennummer des Zertifikats. Dann kann das Zertifikat
# per OpenSSL geprueft werden. Wird vermutlich nur in Serverzertifikaten und
# durch Netscape-Browser unterstuetzt
# Zurueckgegeben wird eine dezimale 0 oder 1
# nsRevocationUrl       = cgi/non-CA-rev.cgi?

# Nur gueltig in CA-Zertifikaten. Bedeutung nicht ganz klar.
# nsCaRevocationUrl     = cgi/CA-rev.cgi?

# Wird verwendet, um einem Benutzer die Erneuerung seines Zertifikats zu
# erleichtern. Ueblicherweise steckt dahinter ein CGI-Script, auf das per
# HTTP GET in der Form ../foo.cgi?aaaa zugegriffen wird. "aaaa" ist wieder
# Seriennummer. Zurueckgegeben werden kann ein Antrags-Formular zur Erneuerung
# des Zertifikats.
# nsRenewalUrl          = cgi/check-renw.cgi?

#####################################################################

[ SCA_ext ]

# basicConstraints      = critical, CA:FALSE
keyUsage                = digitalSignature, keyEncipherment
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid,issuer:always
subjectAltName          = email:copy
issuerAltName           = issuer:copy
crlDistributionPoints   = URI:http://mystic.pca.dfn.de/SCA.crl
nsCertType              = server
nsBaseUrl               = https://mystic.pca.dfn.de/
nsCaPolicyUrl           = http://www.pca.dfn.de/dfnpca/policy/wwwpolicy.html
```

```
nsComment                = This certificate was issued by a Server CA
nsRevocationUrl          = cgi/non-CA-rev.cgi?
# nsRenewalUrl           = cgi/check-renw.cgi?

####################################################################

[ UCA_ext ]

# basicConstraints       = critical, CA:FALSE
keyUsage                 = digitalSignature, keyEncipherment, keyAgreement
subjectKeyIdentifier     = hash
authorityKeyIdentifier   = keyid,issuer:always
subjectAltName           = email:copy
issuerAltName            = issuer:copy
crlDistributionPoints    = URI:http://mystic.pca.dfn.de/UCA.crl
nsCertType               = client, email
nsBaseUrl                = https://mystic.pca.dfn.de/
nsCaPolicyUrl            = http://www.pca.dfn.de/dfnpca/policy/wwwpolicy.html
nsComment                = This certificate was issued by a User CA
nsRevocationUrl          = cgi/non-CA-rev.cgi?
# nsRenewalUrl           = cgi/check-renw.cgi?

####################################################################

[ v3_ca ]

basicConstraints         = critical, CA:TRUE
subjectKeyIdentifier     = hash
authorityKeyIdentifier   = keyid:always,issuer:always
keyUsage                 = cRLSign, keyCertSign
nsCertType               = sslCA, emailCA, objCA
subjectAltName           = email:copy
issuerAltName            = issuer:copy
crlDistributionPoints    = URI:http://mystic.pca.dfn.de/PCA.crl
nsBaseUrl                = https://mystic.pca.dfn.de/
nsCaPolicyUrl            = http://www.pca.dfn.de/dfnpca/policy/wwwpolicy.html
nsComment                = This certificate is a Root CA Certificate



# RAW DER hex encoding of an extension: beware experts only!
# 1.2.3.5=RAW:02:03
# You can even override a supported extension:
# basicConstraints       = critical, RAW:30:03:01:01:FF

####################################################################

[ crl_ext ]

# CRL extensions.
```

```
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.

issuerAltName            = issuer:copy
authorityKeyIdentifier   = keyid:always,issuer:always
```

# B Aufrufparameter und Optionen von openssl

## B.1 openssl

```
Standard commands
asn1parse      ca             ciphers        crl            crl2pkcs7
dgst           dh             dhparam        dsa            dsaparam
enc            errstr         gendh          gendsa         genrsa
nseq           passwd         pkcs12         pkcs7          pkcs8
req            rsa            s_client       s_server       s_time
sess_id        smime          speed          spkac          verify
version        x509

Message Digest commands (see the 'dgst' command for more details)
md2            md5            mdc2           rmd160         sha
sha1

Cipher commands (see the 'enc' command for more details)
base64         bf             bf-cbc         bf-cfb         bf-ecb
bf-ofb         cast           cast-cbc       cast5-cbc      cast5-cfb
cast5-ecb      cast5-ofb      des            des-cbc        des-cfb
des-ecb        des-ede        des-ede-cbc    des-ede-cfb    des-ede-ofb
des-ede3       des-ede3-cbc   des-ede3-cfb   des-ede3-ofb   des-ofb
des3           desx           idea           idea-cbc       idea-cfb
idea-ecb       idea-ofb       rc2            rc2-40-cbc     rc2-64-cbc
rc2-cbc        rc2-cfb        rc2-ecb        rc2-ofb        rc4
rc4-40         rc5            rc5-cbc        rc5-cfb        rc5-ecb
rc5-ofb
```

## B.2 asn1parse

```
asn1parse [options] <infile
where options are
 -inform arg   input format - one of DER TXT PEM
 -in arg       input file
 -out arg      output file
 -noout arg    don't produce any output
 -offset arg   offset into file
 -length arg   length of section in file
 -i            indent entries
 -oid file     file of extra oid definitions
 -strparse offset
```

```
              a series of these can be used to 'dig' into multiple
              ASN1 blob wrappings
 -out filename output DER encoding to file
```

## B.3 ca

```
usage: ca args

 -verbose        - Talk alot while doing things
 -config file    - A config file
 -name arg       - The particular CA definition to use
 -gencrl         - Generate a new CRL
 -crldays days   - Days is when the next CRL is due
 -crlhours hours - Hours is when the next CRL is due
 -startdate YYMMDDHHMMSSZ  - certificate validity notBefore
 -enddate YYMMDDHHMMSSZ    - certificate validity notAfter (overrides -days)
 -days arg       - number of days to certify the certificate for
 -md arg         - md to use, one of md2, md5, sha or sha1
 -policy arg     - The CA 'policy' to support
 -keyfile arg    - PEM private key file
 -key arg        - key to decode the private key if it is encrypted
 -cert file      - The CA certificate
 -in file        - The input PEM encoded certificate request(s)
 -out file       - Where to put the output file(s)
 -outdir dir     - Where to put output certificates
 -infiles ....   - The last argument, requests to process
 -spkac file     - File contains DN and signed public key and challenge
 -ss_cert file   - File contains a self signed cert to sign
 -preserveDN     - Don't re-order the DN
 -batch          - Don't ask questions
 -msie_hack      - msie modifications to handle all those universal strings
 -revoke file    - Revoke a certificate (given in file)
 -extensions ..  - Extension section (override value in config file)
 -crlexts ..     - CRL extension section (override value in config file)
```

## B.4 ciphers

Nach Aufruf werden die von OpenSSL unterstützten Cipher-Suites angezeigt.

## B.5 crl

```
usage: crl args

 -inform arg     - input format - default PEM (DER or PEM)
 -outform arg    - output format - default PEM
 -text           - print out a text format version
 -in arg         - input file - default stdin
```

```
 -out arg        - output file - default stdout
 -hash           - print hash value
 -issuer         - print issuer DN
 -lastupdate     - lastUpdate field
 -nextupdate     - nextUpdate field
 -noout          - no CRL output
 -CAfile  name   - verify CRL using certificates in file "name"
 -CApath  dir    - verify CRL using certificates in "dir"
```

## B.6 crl2pkcs7

```
crl2pkcs7 [options] <infile >outfile
where options are
 -inform arg    input format - DER or PEM
 -outform arg   output format - DER or PEM
 -in arg        input file
 -out arg       output file
 -certfile arg  certificates file of chain to a trusted CA
                (can be used more than once)
 -nocrl         no crl to load, just certs from '-certfile'
```

## B.7 dgst

```
options are
-c   to output the digest with separating colons
-d   to output debug info
-md5 to use the md5 message digest algorithm (default)
-md2 to use the md2 message digest algorithm
-sha1 to use the sha1 message digest algorithm
-sha to use the sha message digest algorithm
-mdc2 to use the mdc2 message digest algorithm
-ripemd160 to use the ripemd160 message digest algorithm
```

## B.8 dh

```
dh [options] <infile >outfile
where options are
 -inform arg   input format - one of DER PEM
 -outform arg  output format - one of DER PEM
 -in arg       input file
 -out arg      output file
 -check        check the DH parameters
 -text         print a text form of the DH parameters
 -C            Output C code
 -noout        no output
```

## B.9 dhparam

```
dhparam [options] [numbits]
where options are
 -inform arg   input format - one of DER PEM
 -outform arg  output format - one of DER PEM
 -in arg       input file
 -out arg      output file
 -check        check the DH parameters
 -text         print a text form of the DH parameters
 -C            Output C code
 -2            generate parameters using  2 as the generator value
 -5            generate parameters using  5 as the generator value
 numbits       number of bits in to generate (default 512)
 -rand file:file:...
               - load the file (or the files in the directory) into
               the random number generator
 -noout        no output
```

## B.10 dsa

```
dsa [options] <infile >outfile
where options are
 -inform arg     input format - DER or PEM
 -outform arg    output format - DER or PEM
 -in arg         input file
 -passin arg     input file pass phrase
 -out arg        output file
 -passout arg    output file pass phrase
 -des            encrypt PEM output with cbc des
 -des3           encrypt PEM output with ede cbc des using 168 bit key
 -idea           encrypt PEM output with cbc idea
 -text           print the key in text
 -noout          don't print key out
 -modulus        print the DSA public value
```

## B.11 dsaparam

```
dsaparam [options] [bits] <infile >outfile
where options are
 -inform arg   input format - DER or PEM
 -outform arg  output format - DER or PEM
 -in arg       input file
 -out arg      output file
 -text         print the key in text
 -C            Output C code
 -noout        no output
 -rand         files to use for random number input
```

```
 number         number of bits to use for generating private key
```

## B.12 enc

```
options are
-in <file>     input file
-out <file>    output fileencrypt
-e             encrypt
-d             decrypt
-a/-base64     base64 encode/decode, depending on encryption flag
-k             key is the next argument
-kfile         key is the first line of the file argument
-K/-iv         key/iv in hex is the next argument
-[pP]          print the iv/key (then exit if -P)
-bufsize <n>   buffer size
Cipher Types
des     : 56 bit key DES encryption
des_ede :112 bit key ede DES encryption
des_ede3:168 bit key ede DES encryption
idea    :128 bit key IDEA encryption
rc2     :128 bit key RC2 encryption
bf      :128 bit key Blowfish encryption
 -rc4   :128 bit key RC4 encryption
 -des-ecb      -des-cbc      -des-cfb      -des-ofb      -des  (des-cbc)
 -des-ede      -des-ede-cbc  -des-ede-cfb  -des-ede-ofb  -desx -none
 -des-ede3     -des-ede3-cbc -des-ede3-cfb -des-ede3-ofb -des3 (des-ede3-cbc)
 -idea-ecb     -idea-cbc     -idea-cfb     -idea-ofb     -idea (idea-cbc)
 -rc2-ecb      -rc2-cbc      -rc2-cfb      -rc2-ofb      -rc2  (rc2-cbc)
 -bf-ecb       -bf-cbc       -bf-cfb       -bf-ofb       -bf   (bf-cbc)
 -cast5-ecb    -cast5-cbc    -cast5-cfb    -cast5-ofb    -cast (cast5-cbc)
 -rc5-ecb      -rc5-cbc      -rc5-cfb      -rc5-ofb      -rc5  (rc5-cbc)
```

## B.13 errstr

```
usage: errstr [-stats] <errno> ...
```

## B.14 gendh

```
usage: gendh [args] [numbits]
 -out file - output the key to 'file
 -2    use 2 as the generator value
 -5    use 5 as the generator value
 -rand file:file:...
           - load the file (or the files in the directory) into
             the random number generator
```

## B.15 gendsa

```
usage: gendsa [args] dsaparam-file
 -out file - output the key to 'file'
 -des      - encrypt the generated key with DES in cbc mode
 -des3     - encrypt the generated key with DES in ede cbc mode (168 bit key)
 -idea     - encrypt the generated key with IDEA in cbc mode
 -rand file:file:...
           - load the file (or the files in the directory) into
             the random number generator
 dsaparam-file
           - a DSA parameter file as generated by the dsaparam command
```

## B.16 genrsa

```
usage: genrsa [args] [numbits]
 -des            encrypt the generated key with DES in cbc mode
 -des3           encrypt the generated key with DES in ede cbc mode (168 bit key)
 -idea           encrypt the generated key with IDEA in cbc mode
 -out file       output the key to 'file
 -passout arg    output file pass phrase
 -f4             use F4 (0x10001) for the E value
 -3              use 3 for the E value
 -rand file:file:...
                 load the file (or the files in the directory) into
                 the random number generator
```

## B.17 nseq

```
Netscape certificate sequence utility
Usage nseq [options]
where options are
-in file  input file
-out file output file
-toseq    output NS Sequence file
```

## B.18 passwd

```
Usage: passwd [options] [passwords]
where options are
-crypt             standard Unix password algorithm (default)
-apr1              MD5-based password algorithm
-salt string       use provided salt
-in file           read passwords from file
-stdin             read passwords from stdin
-quiet             no warnings
```

```
-table            format output as table
-reverse          switch table columns
```

## B.19 pkcs12

```
Usage: pkcs12 [options]
where options are
-export       output PKCS12 file
-chain        add certificate chain
-inkey file   private key if not infile
-certfile f   add all certs in f
-name "name"  use name as friendly name
-caname "nm"  use nm as CA friendly name (can be used more than once).
-in  infile   input filename
-out outfile  output filename
-noout        don't output anything, just verify.
-nomacver     don't verify MAC.
-nocerts      don't output certificates.
-clcerts      only output client certificates.
-cacerts      only output CA certificates.
-nokeys       don't output private keys.
-info         give info about PKCS#12 structure.
-des          encrypt private keys with DES
-des3         encrypt private keys with triple DES (default)
-idea         encrypt private keys with idea
-nodes        don't encrypt private keys
-noiter       don't use encryption iteration
-maciter      use MAC iteration
-twopass      separate MAC, encryption passwords
-descert      encrypt PKCS#12 certificates with triple DES (default RC2-40)
-certpbe alg  specify certificate PBE algorithm (default RC2-40)
-keypbe alg   specify private key PBE algorithm (default 3DES)
-keyex        set MS key exchange type
-keysig       set MS key signature type
-password p   set import/export password (NOT RECOMMENDED)
-passin p     input file pass phrase
-passout p    output file pass phrase
-rand file:file:...
              load the file (or the files in the directory) into
              the random number generator
```

## B.20 pkcs7

```
pkcs7 [options] <infile >outfile
where options are
 -inform arg   input format - DER or PEM
 -outform arg  output format - DER or PEM
 -in arg       input file
```

```
 -out arg      output file
 -print_certs  print any certs or crl in the input
 -text         print full details of certificates
 -noout        don't output encoded data
```

## B.21 pkcs8

```
Usage pkcs8 [options]
where options are
-in file        input file
-inform X       input format (DER or PEM)
-passin arg     input file pass phrase
-envpassin arg  environment variable containing input file pass phrase
-outform X      output format (DER or PEM)
-out file       output file
-passout arg    output file pass phrase
-topk8          output PKCS8 file
-nooct          use (nonstandard) no octet format
-embed          use (nonstandard) embedded DSA parameters format
-nsdb           use (nonstandard) DSA Netscape DB format
-noiter         use 1 as iteration count
-nocrypt        use or expect unencrypted private key
-v2 alg         use PKCS#5 v2.0 and cipher "alg"
-v1 obj         use PKCS#5 v1.5 and cipher "alg"
```

## B.22 req

```
req [options] <infile >outfile
where options  are
 -inform arg    input format - DER or PEM
 -outform arg   output format - DER or PEM
 -in arg        input file
 -out arg       output file
 -text          text form of request
 -noout         do not output REQ
 -verify        verify signature on REQ
 -modulus       RSA modulus
 -nodes         don't encrypt the output key
 -key file      use the private key contained in file
 -keyform arg   key file format
 -keyout arg    file to send the key to
 -newkey rsa:bits generate a new RSA key of 'bits' in size
 -newkey dsa:file generate a new DSA key, parameters taken from CA in 'file'
 -[digest]      Digest to sign with (md5, sha1, md2, mdc2)
 -config file   request template file.
 -new           new request.
 -x509          output a x509 structure instead of a cert. req.
 -days          number of days a x509 generated by -x509 is valid for.
```

```
 -newhdr        output "NEW" in the header lines
 -asn1-kludge   Output the 'request' in a format that is wrong but some CA's
                have been reported as requiring
 -extensions .. specify certificate extension section (override value in config file)
 -reqexts ..    specify request extension section (override value in config file)
```

## B.23 rsa

```
rsa [options] <infile >outfile
where options are
 -inform arg     input format - one of DER NET PEM
 -outform arg    output format - one of DER NET PEM
 -in arg         input file
 -passin arg     input file pass phrase
 -in arg         input file
 -out arg        output file
 -passout arg    output file pass phrase
 -des            encrypt PEM output with cbc des
 -des3           encrypt PEM output with ede cbc des using 168 bit key
 -idea           encrypt PEM output with cbc idea
 -text           print the key in text
 -noout          don't print key out
 -modulus        print the RSA key modulus
 -check          verify key consistency
 -pubin          expect a public key in input file
 -pubout         output a public key
```

## B.24 s_client

```
usage: s_client args

 -host host     - use -connect instead
 -port port     - use -connect instead
 -connect host:port - who to connect to (default is localhost:4433)
 -verify arg   - turn on peer certificate verification
 -cert arg     - certificate file to use, PEM format assumed
 -key arg      - Private key file to use, PEM format assumed, in cert file if
                 not specified but cert file is.
 -CApath arg   - PEM format directory of CA's
 -CAfile arg   - PEM format file of CA's
 -reconnect    - Drop and re-make the connection with the same Session-ID
 -pause        - sleep(1) after each read(2) and write(2) system call
 -showcerts    - show all certificates in the chain
 -debug        - extra output
 -nbio_test    - more ssl protocol testing
 -state        - print the 'ssl' states
 -nbio         - Run with non-blocking IO
 -crlf         - convert LF from terminal into CRLF
```

```
 -quiet        - no s_client output
 -ssl2         - just use SSLv2
 -ssl3         - just use SSLv3
 -tls1         - just use TLSv1
 -no_tls1/-no_ssl3/-no_ssl2 - turn off that protocol
 -bugs         - Switch on all SSL implementation bug workarounds
 -cipher       - preferred cipher to use, use the 'openssl ciphers'
                 command to see what is available
```

## B.25 s_server

```
usage: s_server [args ...]

 -accept arg   - port to accept on (default is 4433)
 -context arg  - set session ID context
 -verify arg   - turn on peer certificate verification
 -Verify arg   - turn on peer certificate verification, must have a cert.
 -cert arg     - certificate file to use, PEM format assumed
                 (default is server.pem)
 -key arg      - Private Key file to use, PEM format assumed, in cert file if
                 not specified (default is server.pem)
 -dcert arg    - second certificate file to use (usually for DSA)
 -dkey arg     - second private key file to use (usually for DSA)
 -dhparam arg  - DH parameter file to use, in cert file if not specified
                 or a default set of parameters is used
 -nbio         - Run with non-blocking IO
 -nbio_test    - test with the non-blocking test bio
 -crlf         - convert LF from terminal into CRLF
 -debug        - Print more output
 -state        - Print the SSL states
 -CApath arg   - PEM format directory of CA's
 -CAfile arg   - PEM format file of CA's
 -nocert       - Don't use any certificates (Anon-DH)
 -cipher arg   - play with 'openssl ciphers' to see what goes here
 -quiet        - No server output
 -no_tmp_rsa   - Do not generate a tmp RSA key
 -ssl2         - Just talk SSLv2
 -ssl3         - Just talk SSLv3
 -tls1         - Just talk TLSv1
 -no_ssl2      - Just disable SSLv2
 -no_ssl3      - Just disable SSLv3
 -no_tls1      - Just disable TLSv1
 -no_dhe       - Disable ephemeral DH
 -bugs         - Turn on SSL bug compatibility
 -www          - Respond to a 'GET /' with a status page
 -WWW          - Respond to a 'GET /<path> HTTP/1.0' with file ./<path>
```

## B.26 s_time

```
usage: s_time <args>

-connect host:port - host:port to connect to (default is localhost:4433)
-nbio          - Run with non-blocking IO
-ssl2          - Just use SSLv2
-ssl3          - Just use SSLv3
-bugs          - Turn on SSL bug compatibility
-new           - Just time new connections
-reuse         - Just time connection reuse
-www page      - Retrieve 'page' from the site
-time arg      - max number of seconds to collect data, default 30
-verify arg    - turn on peer certificate verification, arg == depth
-cert arg      - certificate file to use, PEM format assumed
-key arg       - RSA file to use, PEM format assumed, key is in cert file
                 file if not specified by this option
-CApath arg    - PEM format directory of CA's
-CAfile arg    - PEM format file of CA's
-cipher        - preferred cipher to use, play with 'openssl ciphers'
```

## B.27 sess_id

```
usage: sess_id args

 -inform arg     - input format - default PEM (DER or PEM)
 -outform arg    - output format - default PEM
 -in arg         - input file - default stdin
 -out arg        - output file - default stdout
 -text           - print ssl session id details
 -cert           - output certificate
 -noout          - no CRL output
 -context arg    - set the session ID context
```

## B.28 smime

```
Usage smime [options] cert.pem ...
where options are
-encrypt       encrypt message
-decrypt       decrypt encrypted message
-sign          sign message
-verify        verify signed message
-pk7out        output PKCS#7 structure
-des3          encrypt with triple DES
-des           encrypt with DES
-rc2-40        encrypt with RC2-40 (default)
-rc2-64        encrypt with RC2-64
-rc2-128       encrypt with RC2-128
```

```
-nointern      don't search certificates in message for signer
-nosigs        don't verify message signature
-noverify      don't verify signers certificate
-nocerts       don't include signers certificate when signing
-nodetach      use opaque signing
-noattr        don't include any signed attributes
-binary        don't translate message to text
-certfile file other certificates file
-signer file   signer certificate file
-recip  file   recipient certificate file for decryption
-in file       input file
-inkey file    input private key (if not signer or recipient)
-out file      output file
-to addr       to address
-from ad       from address
-subject s     subject
-text          include or delete text MIME headers
-CApath dir    trusted certificates directory
-CAfile file   trusted certificates file
-rand file:file:...
               load the file (or the files in the directory) into
               the random number generator
cert.pem       recipient certificate(s) for encryption
```

## B.29 speed

```
bad value, pick one of
md2      mdc2   md5      hmac      sha1    rmd160
idea-cbc rc2-cbc  rc5-cbc  bf-cbc
des-cbc  des-ede3 rc4
rsa512   rsa1024  rsa2048  rsa4096

dsa512   dsa1024  dsa2048
idea     rc2      des      rsa    blowfish
```

## B.30 spkac

```
spkac [options]
where options are
 -in arg        input file
 -out arg       output file
 -key arg       create SPKAC using private key
 -passin arg    input file pass phrase
 -challenge arg challenge string
 -spkac arg     alternative SPKAC name
 -noout         don't print SPKAC
 -pubkey        output public key
 -verify        verify SPKAC signature
```

## B.31 verify

```
usage: verify [-verbose] [-CApath path] [-CAfile file] cert1 cert2 ...
recognized usages:
        sslclient       SSL client
        sslserver       SSL server
        nssslserver     Netscape SSL server
        smimesign       S/MIME signing
        smimeencrypt    S/MIME encryption
        crlsign         CRL signing
```

## B.32 version

```
usage:version -[avbofp]
```

## B.33 x509

```
usage: x509 args
 -inform arg     - input format - default PEM (one of DER, NET or PEM)
 -outform arg    - output format - default PEM (one of DER, NET or PEM)
 -keyform arg    - private key format - default PEM
 -CAform arg     - CA format - default PEM
 -CAkeyform arg  - CA key format - default PEM
 -in arg         - input file - default stdin
 -out arg        - output file - default stdout
 -passin arg     - private key password
 -envpassin arg  - read private key password from environment variable "arg"
 -serial         - print serial number value
 -hash           - print hash value
 -subject        - print subject DN
 -issuer         - print issuer DN
 -startdate      - notBefore field
 -enddate        - notAfter field
 -purpose        - print out certificate purposes
 -dates          - both Before and After dates
 -modulus        - print the RSA key modulus
 -pubkey         - output the public key
 -fingerprint    - print the certificate fingerprint
 -alias          - output certificate alias
 -noout          - no certificate output
 -trustout       - output a "trusted" certificate
 -clrtrust       - clear all trusted purposes
 -clrreject      - clear all rejected purposes
 -addtrust arg   - trust certificate for a given purpose
 -addreject arg  - reject certificate for a given purpose
```

```
-setalias arg   - set certificate alias
-days arg       - How long till expiry of a signed certificate - def 30 days
-signkey arg    - self sign cert with arg
-x509toreq      - output a certification request object
-req            - input is a certificate request, sign and output.
-CA arg         - set the CA certificate, must be PEM format.
-CAkey arg      - set the CA key, must be PEM format
                  missing, it is assumed to be in the CA file.
-CAcreateserial - create serial number file if it does not exist
-CAserial       - serial file
-text           - print the certificate in text form
-C              - print out C code forms
-md2/-md5/-sha1/-mdc2 - digest to use
-extfile        - configuration file with X509V3 extensions to add
-extensions     - section from config file with X509V3 extensions to add
```

# C Über dieses Dokument

### Das OpenSSL Handbuch

Dieses Dokument wurde zusammengestellt von den Mitarbeitern der *DFN-PCA* <http://www.pca.dfn.de/dfnpca/> mit maßgeblicher Unterstützung durch Lars Weber (*3weber@informatik.uni-hamburg.de*).

# D Links zu der dokumentierten Software

- **OpenSSL**: *http://www.openssl.org/*
  (*DFN-PCA Mirror* <ftp://ftp.pca.dfn.de/pub/tools/net/openssl/>)
- Obsolet: **SSLeay**: *http://www.ssleay.org/*
  (*DFN-PCA Mirror* <ftp://ftp.pca.dfn.de/pub/tools/net/ssleay/>)
- Obsolet: **pkcs12**: *http://www.drh-consultancy.demon.co.uk/pkcs12faq.html*
- **pfx**: *http://www.drh-consultancy.demon.co.uk/pkcs12faq.html*
- Obsolet: **ca-fix**: *http://www.drh-consultancy.demon.co.uk/ca-fix.html*
- **Apache-SSL**: *http://www.apache-ssl.org/*
  (*DFN-PCA Mirror* <ftp://ftp.pca.dfn.de/pub/tools/net/sslapache/>)
- **mod_ssl**: *http://www.modssl.org/*
  (*DFN-PCA Mirror* <ftp://ftp.pca.dfn.de/pub/tools/net/mod_ssl/>)

## D.1 Browser-Relevantes

- Farrell McKay's **Fortify** - Starke Kryptographie für Netscape-Browser:
  - *http://www.fortify.net/*
    (*DFN-PCA Mirror* <ftp://ftp.pca.dfn.de/pub/tools/net/Fortify/>)
- **Opera** - Ein neuer Shareware-Browser mit starker Kryptographie:
  - *http://www.operasoftware.com/download.html*

## D.2 Weitere Tools

- Netscape's **Certificate Server Extras**:
  - *http://developer.netscape.com/tech/security/certs/certs.html*
  - (*DFN-PCA Mirror für Solaris* `<http://www.pca.dfn.de/dfnpca/certify/ssl/handbuch/ /solaris.cgi.tar.gz>`)
- Microsoft's **CertMgr**:
  - *http://www.microsoft.com/security/tech/certificates/formats.asp#tools*
  - *http://msdn.microsoft.com/downloads/tools/authcodeie4/authcodeie4.asp?*
  - *http://www.microsoft.com/security/downloads/certinst.exe* (*DFN-PCA Mirror*)
- Peter Gutmann's **dumpasn1**:
  - *http://www.cs.auckland.ac.nz/~pgut001/dumpasn1.c*
  - *http://www.cs.auckland.ac.nz/~pgut001/dumpasn1.cfg*
- GMD's **SECUDE**:
  - *http://www.darmstadt.gmd.de/secude/*
- ...