Das OpenSSL Handbuch

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

Version 1.14 02.03.2000

OpenSSL ist eine frei verfügbare Implementierung des SSL/TLS-Protokolls und bietet zahlreiche Funktionen zur X509-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-0.9.2b`. 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 Installation von OpenSSL-0.9.2b

OpenSSL ist eine frei verfügbare Implementierung des *SSL/TLS-Protokolls* und bietet zahlreiche 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 wird derzeit von einer unabhängigen Gruppe weiterentwickelt.

Das Paket umfaßt mehrere einzelne Applikationen, z.B. zur Erzeugung von Zertifikaten, von Requests, zur Verschlüsselung usw. Es gibt zwei Möglichkeiten, das Paket zu kompilieren. Die erste Möglichkeit faßt die einzelnen Applikationenen zu einem monolithischen Programm zusammen, `openssl`. `openssl` wird dann mit dem Applikationsnamen als Parameter aufgerufen.

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

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-0.9.2b.

2.1 Änderungen und Anpassungen am Paket

2.1.1 Änderungen

In `apps/ca.c`

Wenn die `ca`-Applikation die Umgebungsvariable `OPENSSL_CONF` (früher `SSLEAY_CONF`) auswerten soll, muß die folgende Änderung in `apps/ca.c` ab Zeile 404 vorgenommen werden:

```
404      if (configfile == NULL)
405              {
406              /* We will just use 'buf[0]' as a temporary buffer.  */
```

ersetzen durch

```
404      if (configfile == NULL)
405        configfile = getenv( "OPENSSL_CONF" );
406      if (configfile == NULL)
407        {
408          /* We will just use 'buf[0]' as a temporary buffer.  */
```

2.1.2 Anpassungen

Achtung:
Im Unterschied zum SSLeay-Paket müssen die folgenden Anpassungen *nach* der Konfiguration (siehe Übersetzung als Monolith (2.2) bzw. Übersetzung als Programmsammlung (2.3)) des Pakets erfolgen. Der Übersichtlichkeit wegen stehen die Anpassungen schon an dieser Stelle.

Das OpenSSL-Paket enthält eine Reihe von Perl-Skripten, für die der Pfad angepaßt werden kann. Das ist nicht unbedingt notwendig, da während der Kompilation sämtliche Skripte durch `perl script.pl` aufgerufen werden. Allerdings können im späteren Betrieb einige Skripte durch direkte Angabe des Skriptnamens gestartet werden, so daß das vorangestellte „perl“ entfallen kann.

Die Pfad-Anpassung wird größtenteils durch ein Skript erledigt. Achtung, nur den Pfad angeben, z.B. `/usr/bin` wenn der Pfad `/usr/bin/perl` lautet:

```
perl util/perlpath.pl /neuer/pfad
```

In zwei Skripten muß der Pfad von Hand geändert werden:

- In `apps/der_chop`, Zeile 1 ändern
- In `crypto/bf/asm/bf-686.pl`, Zeile 2 löschen

Um den auf `/usr/local/ssl` voreingestellten Installationspfad zu ändern, müssen die Pfadangaben in einigen Dateien geändert werden. Auch wenn nicht mit `make install` installiert wird, müssen die Pfadangaben angepaßt werden, weil sonst möglicherweise Pfade einkompiliert werden, die nicht gewollt sind. Die Pfadanpassungen erfolgen am einfachsten mit folgendem Befehl:

```
perl util/ssldir.pl /new/ssl/home
```

Alternativ kann die Anpassung von Hand erfolgen:

- Datei `crypto/cryptlib.h`, Zeilen 88, 89, 90 und 91
- Datei `tools/c_rehash`, Zeile 11
- Datei `Makefile.ssl`, Zeile 139
- Datei `util/mk1mf`, Zeile 8

2.2 Übersetzung und Installation als Monolith

Mit

```
perl Configure solaris-sparc-gcc
```

wird das Paket für die Übersetzung konfiguriert. Alternativ dazu bietet sich die Möglichkeit einer automatischen Erkennung mit dem Befehl

```
sh config
```

Nach Abschluß der Konfiguration wird ein Hinweis ausgegeben, daß das OpenSSL-Programm (noch) nicht verfügbar ist:

```
c_rehash: rehashing skipped ('openssl' program not available)
```

Das liegt daran, daß das Konfigurations-Skript versucht Hash-Links auf einige Zertifikate zu setzen, wozu das Programm `openssl` (welches ja erst noch kompiliert werden soll) benötigt wird. Nachdem das Programm übersetzt wurde, kann dieser Vorgang mit dem untenstehenden Befehl („`make rehash`") nachgeholt werden.

Achtung:
Bevor folgende Schritte durchgeführt werden, sollten jetzt die unter Anpassungen (2.1.2) aufgeführten Änderungen vorgenommen werden.

Mit

```
make clean ; make errors ; make
```

wird das Paket übersetzt. Durch `make errors` wird numerischen Fehlercodes ein kurzer, beschreibenender Fehlertext zugeordnet. Nun sollte noch

```
make rehash ; make test
```

aufgerufen werden. Durch das erste Kommando `make rehash` werden die oben erwähnten Hash-Werte in einem Demo-Verzeichnis wiederhergestellt, welche für den folgenden Testlauf gebraucht werden. Mit `make test` werden umfangreiche Tests des Paketes durchgeführt. Erfolgte der Testlauf im Sinne der Tests fehlerfrei, kommt abschließend eine Meldung ähnlich der folgenden:

```
OpenSSL 0.9.2b 22 Mar 1999
built on: Thu Apr  8 12:26:12 MET DST 1999
platform: solaris-sparc-gcc
options:  bn(64,32) md2(int) rc4(ptr,char) des(idx,cisc,16,long) idea(int) blowfish(ptr)
compiler: gcc -O3 -fomit-frame-pointer -mv8 -Wall -DB_ENDIAN
'test' is up to date.
```

Die 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.

Wer etwas mehr Kontrolle über die Installation haben will, kann sich an folgendes Verfahren halten:

Für die Installation muß zunächst ein Zielverzeichnis erzeugt werden, in dem Programme, Bibliotheken usw. installiert werden können. Der Pfad dieses Verzeichnisses sollte günstigerweise mit dem vor der Kompilierung angegebenen übereinstimmen. In diesem Verzeichnis werden dann die Verzeichnisse `bin`, `crl`, `certs`, `lib`, `newcerts`, `private` erzeugt. Bei Bedarf kann auch noch ein Verzeichnis `include` erzeugt werden, in das dann die Header-Dateien der OpenSSL-Bibliotheken kopiert werden. Das ist sinnvoll, wenn die ergänzenden Programme von Steve Henson (siehe pfx (5.1) und pkcs12 (5.2)) eingesetzt bzw. kompiliert werden sollen. Anschließend werden die Dateien kopiert (*im folgenden $(SSLDIR) = Installationspfad von OpenSSL*):

- `cp libcrypto.a libssl.a apps/openssl.cnf $(SSLDIR)/lib/`
- `chmod 644 $(SSLDIR)/lib/*`
- `cp tools/c_* apps/der_chop apps/CA.sh apps/openssl $(SSLDIR)/bin/`
- `chmod 755 $(SSLDIR)/bin/*`
- `cp include/* $(SSLDIR)/include/`
- `e_os.h`, `date.h` und `rsaref.h` werden nicht benötigt und sollten gelöscht werden:
  `rm $(SSLDIR)/include/e_os.h $(SSLDIR)/include/date.h $(SSLDIR)/include/rsaref.h`
- `chmod 644 $(SSLDIR)/include/*`

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

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

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

```
touch $(SSLDIR)/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 (3.7.1)`).

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)`.

2.3 Übersetzung und Installation als Programmsammlung

Mit

```
perl Configure solaris-sparc-gcc
```

wird das Paket für die Übersetzung konfiguriert. Alternativ dazu bietet sich die Möglichkeit einer automatischen Erkennung mit dem Befehl

```
sh config
```

Nach Abschluß der Konfiguration wird ein Hinweis ausgegeben, daß das OpenSSL-Programm (noch) nicht verfügbar ist:

```
c_rehash: rehashing skipped ('openssl' program not available)
```

Das liegt daran, daß das Konfigurations-Skript versucht, Hash-Links auf einige Zertifikate zu setzen, wozu das Programm `openssl` (welches ja erst noch kompiliert werden soll) benötigt wird. Nachdem das Programm übersetzt wurde, kann dieser Vorgang mit untenstehenden Befehl („`make rehash`“) nachgeholt werden.

Achtung:
Bevor folgende Schritte durchgeführt werden, sollten jetzt die unter Anpassungen (2.1.2) aufgeführten Änderungen vorgenommen werden.

Mit

```
make clean ; make errors ; make
```

wird das Paket übersetzt. Durch `make errors` wird numerischen Fehlercodes ein kurzer, den Fehler beschreibenden, Fehlertext zugeordnet. Nun sollte noch

```
make rehash ; make test
```

aufgerufen werden. Durch das erste Kommando `make rehash` werden die oben erwähnten Hash-Werte in einem Demo-Verzeichnis wiederhergestellt, welche für den folgenden Testlauf gebraucht werden. Mit `make test` werden umfangreiche Tests des Paketes durchgeführt. Erfolgte der Testlauf im Sinne der Tests fehlerfrei, kommt abschließend eine Meldung ähnlich der folgenden:

```
OpenSSL 0.9.2b 22 Mar 1999
built on: Thu Apr  8 12:26:12 MET DST 1999
platform: solaris-sparc-gcc
options:  bn(64,32) md2(int) rc4(ptr,char) des(idx,cisc,16,long) idea(int) blowfish(ptr)
compiler: gcc -O3 -fomit-frame-pointer -mv8 -Wall -DB_ENDIAN
‘test’ is up to date.
```

Jetzt ist das komplette Paket übersetzt; die Übersetzung diente aber nur dazu, die Bibliotheken `libssl.a` und `libcrypto.a` im Quellverzeichnis zu erzeugen. Die einzelnen Applikationen werden am einfachsten mit folgendem Shell-Skript im Quell-Verzeichnis übersetzt:

```
#!/bin/sh -x
for i in asn1pars ca ciphers crl crl2p7 dgst dh dsa dsaparam enc errstr gendh \
         gendsa genrsa nseq pkcs7 req rsa sess_id speed verify version x509
do
 gcc -Icrypto -Iinclude -fomit-frame-pointer -mv8 -Wall -o out/$i \
      apps/$i.c apps/apps.c libssl.a libcrypto.a
done
```

Zwei Applikationen sind noch aus Kompatibilitätsgründen mit der OpenSSL-Installation als Monolith umzubenennen:

- `mv out/asn1pars out/asn1parse`
- `mv out/crl2p7 out/crl2pkcs7`

Für die folgenden Applikationen war die Übersetzung mit dieser Methode nicht erfolgreich:

`s_client, s_server, s_time`

Eine Beschreibung der Funktion der einzelnen Applikationen findet sich im Anhang OpenSSL-Parameter (B).

Für die Installation muß zunächst ein Zielverzeichnis erzeugt werden, in dem Programme, Bibliotheken usw. installiert werden können. Der Pfad dieses Verzeichnisses sollte günstigerweise mit dem vor der Kompilierung angegebenen übereinstimmen. In diesem Verzeichnis werden dann die Verzeichnisse `bin`, `crl`, `certs`, `lib`, `newcerts`, `private` erzeugt.

Bei Bedarf kann auch noch ein Verzeichnis `include` erzeugt werden, in das dann die Header-Dateien der OpenSSL-Bibliotheken kopiert werden. Das ist sinnvoll, wenn die ergänzenden Programme von Steve Henson (siehe pfx (5.1) und pkcs12 (5.2)) eingesetzt bzw. kompiliert werden. Anschließend werden die Dateien kopiert (*im folgenden $(SSLDIR) = Installationspfad von OpenSSL*):

- `cp libcrypto.a libssl.a apps/openssl.cnf $(SSLDIR)/lib`

- chmod 644 $(SSLDIR)/lib/*
- cp tools/c_* apps/der_chop apps/CA.sh out/* $(SSLDIR)/bin
- chmod 755 $(SSLDIR)/bin/*
- cp include/* $(SSLDIR)/include
- e_os.h, date.h und rsaref.h werden nicht benötigt und sollten gelöscht werden:

  rm $(SSLDIR)/include/e_os.h $(SSLDIR)/include/date.h $(SSLDIR)/include/rsaref.h
- chmod 644 $(SSLDIR)/include/*

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

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

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

```
touch $(SSLDIR)/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 eins erhöht und das herausgegebene Zertifikat wird in `index.txt` registriert (siehe index.txt (3.7.1)).

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 `openssl.cnf (A)`.

3 Zum Gebrauch von OpenSSL

3.1 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 `$(SSL_DIR)/lib/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/ssl/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 Zertfizierung durch die Applikation `ca` der Abschnitt `ca_default` gewählt.

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 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 (Anm.: Statt der oben erwähnten zwei speziellen Konfigurationsdateien kann das gleiche auch über entsprechende Konfigurationsabschnitte innerhalb einer Datei 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 Zertifikaterweiterungen (*Extensions*) für neue Zertifikate festgelegt sind. In diesem Abschnitt könnten die Extensions fü 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 ebenfalls drei Schlüsselwörter erkannt:

- `req_distinguished_name`
- `attributes`
- `x509_extensions`

`req_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 Privat-Key. Genaueres siehe Erzeugen von Requests (3.5). In dem Bereich, auf den `x509_extensions` verweist, werden die Zertifikaterweiterungen festgelegt, die bei Erzeugung eines selbsignierten Zertifikats (*Wurzel-Zertifikat*) in dieses Zertifikat gebracht werden.

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

3.2 Zufallszahlen mit OpenSSL

Anmerkung:
Die folgenden Angaben beziehen sich auf SSLeay-0.8.1. Möglicherweise sind sie auch noch für OpenSSL-0.9.5-dev gültig.

OpenSSL bringt einen eigenen Zufallszahlen-Generator (ZZG) mit. Im folgenden ein paar erläuternde Worte zur Funktion des ZZG. Vor der ersten Schlüsselerzeugung mit OpenSSL sollte in jedem Fall der ZZG initialisiert werden. Andernfalls ist die zur Schlüsselerzeugung heranziehbare Zufalls-Datenmenge sehr klein. Die Initialisierung kann erfolgen, indem eine beliebige Datei, die als Zufallsdaten brauchbare Daten enthält, unter dem für den ZZG-Status vorgesehenen Namen abgespeichert wird. Vorgesehener Name heißt, ein vor der Kompilierung festgelegter Name (`$(SSLSRC)/e_os.h`, Zeile 187), ein über eine Umgebungsvariable `$RANDFILE` festgelegter Name oder ein über die Konfigurationsdatei festgelegter Name. Wird allerdings eine Schlüsselerzeugung mit dem Kommando „`openssl genrsa`" durchgeführt, muß vorher `$RANDFILE` gesetzt sein, da das Kommando die Konfigurationsdatei nichtauswertet. Es wird dann die in `e_os.h` festgelegte Default-Datei angelegt.

Die Initialisierung des ZZG-Status könnte also z.B. durch nachstehendes Kommando erfolgen:

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

Beim ersten Zugriff auf die so initialisierte Status-Datei durch `openssl` wird die Datei nach der ersten Schlüsselerzeugung auf eine Länge von 1024 Byte begrenzt, unabhängig davon, ob weniger oder mehr Byte zur Initialisierung verwendet wurden. Jede neue Schlüsselerzeugung beeinflußt den ZZG-Status. Der Status und neue Zufallsdaten werden in Blöcke zu 16 Byte zerlegt und diese dann anschließend paarweise durch den MD5-Hash-Algorithmus geleitet. Das Ergebnis wird in einer Variable `md` zwischengespeichert. Dieser Zwischenspeicher (`md`) wird dann XOR mit dem ZZG-Status an derselben Stelle, aus der vorher die 16 Byte entnommen worden, verknüpft. Bei jedem schreibenden oder lesenden Zugriff auf den ZZG-Status wird ein Zeiger inkrementiert, so daß nicht dieselben 16 Byte des ZZG-Status verwendet werden. Durch dieses Vorgehen sollen Rückschlüsse vom ZZG-Status auf die verwendeten Zufallsdaten unmöglich gemacht werden.

Die Entnahme von Zufallszahlen aus dem ZZG geschieht folgendermaßen: Je 8 Byte werden aus dem ZZG-Status entnommen und durch den MD5-Hash-Algorithmus geleitet. Von der MD5-Ausgabe werden die ersten 8 Bytes als Zufallszahl zurückgegeben und die zweiten 8 Bytes über XOR in den ZZG-Status gebracht. Nachdem die benötigten Zufallsdaten erzeugt worden sind, werden der Wert des Zeigers auf die Position im ZZG-Status sowie der aktuelle Wert von `md` wieder durch den MD5-Algorithmus geleitet und das Ergebnis wird in `md` gehalten. Werden Schlüsselpaare über `genrsa` erzeugt, kann zur Initialisierung und auch bei jedem weiteren Aufruf mit der Option `-rand file1:file2:...` eine Liste von Dateien angegeben werden, die alle als Zufallsdaten Verwendung finden.

Wird ein Schlüsselpaar erzeugt, ohne daß der Status existiert, gibt das Programm die Meldung *unable to load 'random state'* aus. Werden auch keine Initialisierungs-Daten mitgegeben, meldet das Programm ferner *warning, not much extra random data, consider using the -rand option.* In diesem Fall werden die System-Funktionen `getpid()`, `getuid()`, `time()` sowie `stat()` auf die nicht vorhandene Status-Datei angewendet. Eine Schlüsselerzeugung findet also statt, und die entsprechend „zufälligen" Daten der obigen Systemfunktionen werden dann zur Initialisierung des Status' herangezogen.

3.3 Zertifizieren mit OpenSSL

OpenSSL in der Version 0.9.2b unterstützt die folgenden *X509.v3-Erweiterungen (Extensions)*:

- *Certificate Standard Extensions*

- Key Usage
- Basic Constraints
- Subject Key Identifier
- Authority Key Identifier
- Subject Alternative Name
- Issuer Alternative Name

- *Netscape Certificate Extensions* `<http://home.netscape.com/eng/security/certs.html>` (proprietär)

Zertifikaterweiterungen steuern den Verwendungszweck von Zertifikaten, sofern die Anwendungssoftware diese Erweiterungen korrekt interpretiert. Üblicherweise werden diese Erweiterungen durch eine (P)CA beim Signieren eines Request in das dann erstellte Zertifikat gebracht. Die Bedeutung der „Certificate 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.

Es ist auch möglich, eigene Erweiterungen zu registrieren (was Netscape mit den „Netscape Certificate 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).

Wegen der besonderen Bedeutung von **Basic Constraints** und **Key Usage** an dieser Stelle noch ein paar Anmerkungen:

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, „*NetscapeCertificate 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. (3.3)) markiert ist. Liegen andererseits mehrere Zertifikate vor (z.B. eins zum Signieren, eins 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 Key Usage (3.3.1)).

Mittels Basic Constraints kann eine Anwendung erkennen, ob es sich 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.

Critical Bit

Die oben aufgeführten Extensions Basic Constraints und Key Usage können als **Critical** markiert werden. Bei korrekter Implementierung einer Anwendung *muß diese eine als Critical markierte Extension interpretieren.* Ist die Anwendung dazu nicht in der Lage, hat sie das Zertifikat, das diese Extension enthält, zurückzuweisen, auch wenn es ansonsten technisch korrekt ist. Allerdings gibt es verschiedene Ansichten über die Verwendung der einzelnen Key Usage Attribute, so daß eine Critical-Markierung nicht ohne „Risiko“ ist.

Zu den Netscape Certificate Extensions ist anzumerken, daß diese nicht Critical gesetzt werden sollten. Ein SSL-Server-Zertifikat, das beispielsweise das Netscape-SSL-Server-Bit (nsCertType) als Critical gesetzt hat, wird von einem Microsoft-Browser (der die Extension nicht kennt) zurückgewiesen. Ein solches Server-Zertifikat würde also eine SSL-Verbindung von Microsoft-Clients zum Server ausschließen.

Im folgenden ein paar weitere Anmerkungen zu den Erweiterungen.

3.3.1 Key Usage

Key Usage wird von OpenSSL-0.9.2b direkt unterstützt. 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* `<ftp://ftp.informatik.uni-hamburg.de/pub/doc/rfc/rfc2459.txt.gz>`. Es stehen neun Werte für das Schlüsselwort `keyUsage` in der Konfigurationsdatei `openssl.cnf` zur Verfügung:

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).

(Für genauere Angaben siehe *RFC 2459* `<ftp://ftp.informatik.uni-hamburg.de/pub/doc/rfc/rfc2459.txt.gz>` sowie die Beispiel-Konfigurationsdatei weiter unten.)

In der Konfigurationsdatei:

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

3.3.2 Basic Constraints

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 PKIX 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-/Benutzerzertifikate heraus. Basic Constraints sollte immer als Critical markiert werden.

Laut *Steve 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...“.*)

(Für genauere Angaben siehe *RFC 2459* `<ftp://ftp.informatik.uni-hamburg.de/pub/doc/rfc/rfc2459.txt.gz>` sowie die Beispiel-Konfigurationsdatei weiter unten.)

In der Konfigurationsdatei:

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

3.3.3 Subject Key Identifier

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

(Für genauere Angaben siehe *RFC 2459* `<ftp://ftp.informatik.uni-hamburg.de/pub/doc/rfc/rfc2459.txt.gz>` sowie die Beispiel-Konfigurationsdatei weiter unten.)

In der Konfigurationsdatei:

```
subjectKeyIdentifier = hash
```

3.3.4 Authority Key Identifier

Diese Erweiterung besteht aus drei Feldern: `keyIdentifier`, `authorityCertIssuer` und `authorityCertSerialNumber`. Laut PKIX 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. PKIX empfiehlt die erste Variante.

(Für genauere Angaben siehe *RFC 2459* `<ftp://ftp.informatik.uni-hamburg.de/pub/doc/rfc/rfc2459.txt.gz>` sowie die Beispiel-Konfigurationsdatei weiter unten.)

In der Konfigurationsdatei:

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

Ist `keyid` gesetzt, wird Subject Key Identifier (siehe oben (3.3.3)) 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, wird alternativ das Issuer-Feld und die Seriennummer kopiert. Ist `issuer` auf `always` gesetzt, wird immer das Issuer-Feld und die Seriennummer kopiert.

3.3.5 Subject Alternative Name

Diese Erweiterung 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), jeweils „beliebig" viele, möglich. Diese können auch beliebig kombiniert werden.

(Für genauere Angaben siehe *RFC 2459* `<ftp://ftp.informatik.uni-hamburg.de/pub/doc/rfc/rfc2459.txt.gz>` sowie die Beispiel-Konfigurationsdatei weiter unten.)

In der Konfigurationsdatei:

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

3.3.6 Issuer Alternative Name

Diese Extensions ist wie Subject Alternative Name (3.3.5) 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* `<ftp://ftp.informatik.uni-hamburg.de/pub/doc/rfc/rfc2459.txt.gz>` 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]>
```

3.3.7 Netscape Certificate Extensions

Zu den Netscape Certificate Extensions 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-0.9.2b 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`)

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

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:

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.

3.4 Erzeugen eines Root-CA-Zertifikats

Die oben 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. 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
   ```

   Diese und die unter 1. aufgeführten Einträge müssen in dem beim Schlüsselwort `x509_extensions` des Request-Abschnitts genannten Bereich gemacht werden, also z.B. im Bereich `[ v3_ca ]`.

3. Initialisieren des Zufallszahlen-Status, z.B.:

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

4. Erzeugen eines 2048-Bit-Schlüssels:

   Spätestens vor Aufruf dieses Befehls muß `$RANDFILE` gesetzt sein, sonst findet das folgende Kommando die Datei mit dem Zufallszahlen-Status nicht, und ein einkompilierter Default wird wirksam (siehe Zufallszahlen (3.2)).

   ```
   openssl genrsa -des3 -out $(SSLDIR)/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 $(SSLDIR)/private/CAkey.pem -out ./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
   ```

7. Das Zertifikat an den gewünschten (bzw. an den in der Konfigurationsdatei unter „`certificate=`“ benannten) Ort kopieren, z.B.

   ```
   cp ./CAcert.pem $(SSLDIR)/CAcert.pem
   ```

Das Zertifikat muß jetzt noch in das Verzeichnis `$(SSLDIR)/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 (3.4)).

- `<mv|cp> $(SSLDIR)/CAcert.pem $(SSLDIR)/certs/00.pem`
- `cd $(SSLDIR)/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 `$(SSLDIR)/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 Issuer-Feldes (also Distinguished Name und evtl. Email-Adresse des Herausgebers) des Zertifikats gebildet.

Anmerkung:
Bis auf das nach obigem Verfahren erzeugte Root-Zertifikat finden sich erzeugte Zertifikate (zusätzlich zu dem bei `-out` angegebenen Namen) unter `$(SSLDIR)/newcerts/`*xx*`.pem`. Das *xx* steht für die Seriennummer des neuen Zertifikats. Alle neuen Zertifikate sollten von `$(SSLDIR)/newcerts/` nach `$(SSLDIR)/certs/` kopiert und anschließend 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.

3.5 Erzeugen von Certificate Requests

Zur Erzeugung eines beliebigen Requests (CA, Server, Client) muß zunächst ein Schlüsselpaar generiert werden. Mit folgendem Befehl wird ein 1024-(512)-Bit Schlüssel erzeugt:

```
openssl genrsa -des3 -out MyKey.pem -rand file1:file2:... 1024 (512)
```

Vorher sollte allerdings die Umgebungsvariable `$RANDFILE` gesetzt und der Zufallszahlen-Status initialisiert worden sein (siehe Zufallszahlen (3.2)). 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. Für die Netscape Navigator/Communicator (NSC)Export-Version gilt im Prinzip die gleiche Beschränkung. Die Schlüssellänge kann aber bis auf 2048 Bit erhöht werden, wenn das Zertifikat extern (z.B. durch `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 (5.1) bzw. PKCS12 (5.2)).

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/ssl/lib/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 (3.3.5) 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.

Der so erzeugte Request (also nur die Datei *MyReq.pem*, nicht der Schlüssel) kann dann auf Diskette kopiert und einer CA zum Signieren (also: Zertifizieren, siehe Abschnitt Signieren (3.6)) 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 (5.1) und PKCS12 (5.2)).

3.6 Signieren von 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
```

Der Wert von *ca_name* steht hier für den gewünschten Abschnitt der Konfigurationsdatei, also z.B. `Client_CA`.

Nach Eingabe des Befehls kommt eine Meldung folgender Art:

```
Using configuration from /usr/local/ssl/lib/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 Mar 18 09:51:41 1999 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 Kompilation angegebenen SSL-Verzeichnis, kann die Angabe im CA-Kommando bei `-config` */pfad/Konfigname* erfolgen. Oder es wird die

Umgebungsvariable `OPENSSL_CONF` (aber nur nachdem die Anpassungen (2.1.2) vorgenommen wurden) 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 Request muß unbedingt sichergestellt sein, daß die richtigen Extensions gesetzt sind bzw. der richtige Abschnitt in der KD gewählt wird (siehe Zertifizieren (3.6) und Anhang `openssl.cnf (A)`).

Sind in der KD keine anderen Angaben gemacht worden, befindet sich das neu erstellte Zertifikat in der bei `-out` angegebenen Datei. Außerdem wird im Verzeichnis `$(SSLDIR)/newcerts` eine Kopie abgelegt. Der Name der Kopie setzt sich aus der Seriennummer des Zertifikats und der Endung `.pem` zusammen, also z.B. `0a.pem`. Die Seriennummer des gerade herausgegebenen Zertifikats findet sich im Zertifikat selber (`openssl x509 -in myCert.pem -serial`) oder in der Datei `$(SSLDIR)/serial.old`. Die Kopie muß dann nach `$(SSLDIR)/certs` kopiert (wird stattdessen das Original genommen, muß diese umbenannt werden) und über ihren Hash-Wert verlinkt werden:

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

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

3.7 Certificate Revocation Lists (CRL)

In Abhängigkeit vom Status der Zertifikate, die in der Indexdatei `$(SSLDIR)/index.txt` durch Gültigkeit, Seriennummer und Distinguished Name (DN) repräsentiert werden, kann mit dem Kommando

```
openssl ca -gencrl -outform der -out $(SSLDIR)/crl/crl.der
```

eine Certificate Revocation List (CRL) erzeugt werden. Um ein Zertifikat zurückrufen zu können, muß die Indexdatei mit einem Texteditor geändert werden (siehe Widerruf eines Zertifikats (3.7.1)).

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 (3.7.3)* und *CRL Issuer Alternative Name (3.7.4)*. Die Verwendung dieser Extensions zeichnet (u.a.) eine X.509v2-CRL aus. Werden diese Extensions nicht verwendet (durch Löschen oder Auskommentieren des Schlüsselworts `crl_extensions` in `openssl.cnf`), erzeugt obiges Kommando eine X.509v1-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*“.

3.7.1 Aufbau der Indexdatei `index.txt`

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*
3. Ist in obiger Beispielzeile leer. Die Spalte kann aber ebenfalls einen Ablaufzeitpunkt YYMMDDHHMMSS*Z* enthalten. Ist hier ein Datum eingetragen, entspricht dies dem Widerrufzeitpunkt 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.

3.7.2 Widerruf eines Zertifikats

Soll ein Zertifikat widerrufen werden, muß 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 Widerrufzeitpunkt eingetragen werden.

Dann kann mit dem obigen Kommando `ca -gencrl (3.7)` eine CRL mit dem widerrufenen Zertifikat erzeugt werden.

3.7.3 CRL Authority Key Identifier

Diese Extension besteht aus drei Feldern: `keyIdentifier`, `authorityCertIssuer` und `authorityCertSerialNumber`. Laut PKIX 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.

PKIX empfiehlt die erste Variante.

(Für genauere Angaben siehe *RFC 2459* `<ftp://ftp.informatik.uni-hamburg.de/pub/doc/rfc/rfc2459.txt.gz>` sowie die Beispiel-Konfigurationsdatei weiter unten.)

In der Konfigurationsdatei:

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

3.7.4 CRL Issuer Alternative Name

Diese Extension ist wie Subject Alternative Name (3.3.5) aufgebaut und dient dazu, zusätzliche Information zum Herausgeber in die CRL zu bringen.

(Für genauere Angaben siehe *RFC 2459* `<ftp://ftp.informatik.uni-hamburg.de/pub/doc/rfc/rfc2459.txt.gz>` 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]>
```

4 Testbericht und Anmerkungen

4.1 OpenSSL

Das OpenSSL-Paket funktioniert recht stabil. Allerdings führen schon kleine Schreibfehler zu, auf den ersten Blick nicht sehr verständlichen, Fehlermeldungen folgender Art (die zweite Zeile ist eine Meldung vom Betriebssystem):

```
> openssl x509 -inform der -in cert-6,der -text
cert-6,der: No such file or directory
unable to load certificate
261:error:02001002:system library:fopen:system lib:bss_file.c:271:fopen('cert-6,der','r')
261:error:20074002:BIO routines:FILE_CTRL:system lib:bss_file.c:273:
```

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 Applikation (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.

Wird für die Seriennummer in `serial.txt` die Hex-Zahl `00` eingetragen, steht nach Herausgabe des nächsten (ersten) Zertifikates die Seriennummer in der Datei `serial.txt` auf `5C5C5C5D`... Die Seriennummer des gerade herausgegebenen neuen Zertifikats ist dann `0` (einstellig).

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.

Ansonsten ist die Verwaltung von Zertifikaten und der Betrieb einer CA 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 vom Verzeichnis `newcerts` nach `certs` kopiert werden. Anschließend müssen die Zertifikate über ihren Hash-Wert „verlinkt“ werden.
- Es gibt keinen Automatismus, der die Indexdatei auf abgelaufene Zertifikate überprüft und Einträge entsprechend markiert.
- Soll ein Zertifikat widerrufen werden, muß der entsprechende Eintrag über die Seriennummer und den DN des Herausgebers identifiziert und dann wie unter CRL (3.7) beschrieben verändert werden.

4.2 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 (3.7).

Zertifikate

Mit OpenSSL erzeugte Zertifikate lassen sich in den Netscape Communicator/Navigator (NSC) relativ einfach importieren.

Allerdings lassen sich keine Zertifikate, deren Schlüssellängen größer als 1024 Bit ist, mit Browser-Versionen vor 4.0 verwenden. Ein solcher Browser bricht die Verbindung zu einem Server ab, wenn sich in der Zertifikatkette ein Zertifikat mit einem solchen langen Schlüssel befindet.

Zertifikat im (binären) DER-Format kann von einem HTTP-Server als einer der folgenden MIME-Typen an den Browser geschickt werden:

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

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:

<table>
<tr><th><code>nsCertType</code></th><th>Beschreibung</th><th>Rubrik</th><th>Accept this CA for Certifying ...</th></tr>
<tr><td><code>objCA</code></td><td>CA-Zert. für 0x10-Zert.</td><td>Signers</td><td>... Software-Developers</td></tr>
<tr><td><code>emailCA</code></td><td>S/MIME-CA</td><td>Signers</td><td>... e-mail users</td></tr>
<tr><td><code>sslCA</code></td><td>SSL CA</td><td>Signers</td><td>... network sites</td></tr>
<tr><td><code>reserved</code></td><td>Reserviert</td><td>Signers</td><td rowspan="5">Für <u>alle fünf</u> CertTypes
folgende Auswahl:
... Software-Developers
... network sites
... e-mail users</td></tr>
<tr><td><code>objsign</code></td><td>Zur Objekt-Signierung</td><td>Signers</td></tr>
<tr><td><code>email</code></td><td>S/MIME Benutzer-Zert.</td><td>Signers</td></tr>
<tr><td><code>server</code></td><td>SSL-Server</td><td>Signers</td></tr>
<tr><td><code>client</code></td><td>SSL-Client</td><td>Signers</td></tr>
</table>

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 folgender 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` `sslCA`	SSL-CA SSL-CA	Signers	... network sites ... 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 Steve 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.* 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 in PFX (5.1) bzw. PKCS12 (5.2) 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.)

CRLs

Achtung:
Netscape-Browser akzeptieren (bisher) keine X.509v2 CRLs. Genaueres steht im Abschnitt über CRLs (3.7).

Die im folgenden verwendeten CRLs waren vom Typ X.509v1.

CRL als MIME-Type `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 `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.

4.3 Zertifikate und CRLs mit MS Internet Explorer

Der Internet Explorer (MSIE) unterstützt u.a. die Standard X509v3-Attribute wie Basic Constraints und Key Usage (siehe jedoch Zertifizieren (3.6)). 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 folgende Angaben beziehen sich auf den MSIE 4.0. Es ist dabei zu berücksichtigen, daß sich das Verhalten des Browsers unter Windows NT in Abhängigkeit vom installierten Servicepack ändern kann.

Der MSIE akzeptiert Schlüssellängen von 2048 Bit für CA-Zertifikate. Für Benutzerzertifikate beträgt die max. Schlüssellänge 512 Bit bei der Export-Version des Browsers.

Allerdings lassen sich keine Zertifikate, deren Schlüssellängen größer als 1024 Bit ist, mit Browser-Versionen vor 4.0 verwenden. Ein solcher Browser bricht die Verbindung zu einem Server ab, wenn sich in der Zertifikatkette ein Zertifikat mit einem solchen langen Schlüssel befindet.

CA-Zertifikate können über einen Web-Server als MIME-Type `x-x509-ca-cert` gesendet werden (siehe

Testbericht (4)). Sie werden als CA-Zertifikate für „Netzwerk-Client-“, „Netzwerk-Server-Authentifikation“, „Sichere-E-Mail“ und „Software-Herausgeber“ akzeptiert. Der Benutzer wird dabei durch einen Interaktionmodus geführt, in deren Verlauf das Zertifikat entweder lokal auf der Festplatte gespeichert oder gleich geöffnet werden kann. Wurde das Zertifikat gespeichert, kann es später jederzeit durch einen „Doppelklick“ geladen (akzeptiert) werden. Nach Akzeptieren findet sich das Zertifikat in der Rubrik „Ansicht / Internetoptionen / Inhalt / Agenturen“. Es besteht die Möglichkeit, über Kontrollkästchen die Beschränkungen des Zertifikats einzeln zu (de-)aktivieren.

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. Wenn die Schlüsselerzeugung durch einen Export-MSIE erfolgt, ist die Schlüssellänge auf 512 Bit beschränkt. Durch Austausch einer Krypto-Bibliothek im Windows-System besteht die Möglichkeit, die Schlüssellänge zu erhöhen. Genaueres dazu findet sich auf Hensons *PKCS#12-Seite* `<http://www.drh-consultancy.demon.co.uk/pkcs12faq.html/pkcs-12.html>`. Auf die Methode Schlüsselerzeugung durch den Browser und anschließender Online-Zertifizierung wird hier nicht weiter eingegangen.

Die zweite Möglichkeit besteht darin, wie im Abschnitt Erzeugen von Requests (3.5) beschrieben, einen mit OpenSSL erzeugten Request zu signieren und diesen anschließend als `.p12`-Datei in den Browser zu importieren. Allerdings besteht die Möglichkeit größere Schlüssel zu importieren, wenn der oben erwähnte Austausch einer Krypto-Bibliothek erfolgte. Um das Zertifkat (in Form der `.p12`-Datei) zu laden, muß in der Rubrik „Ansicht / Internetoptionen / Inhalt / Eigene“ eine Dialogbox geöffnet werden, wo über den Button „Importieren“ das Zertifikat importiert werden kann. Die in den Browser gebrachten Zertifikate stehen auch im MS-Mail Programm Outlook-Express (OE) zur Verfügung, wenn die im Zertifikat angegebene E-Mail-Adresse mit der im OE übereinstimmt.

Benutzer-Zertifikate erfahren keinen speziellen Schutz durch den MSIE. Der einzige Schutz besteht im Zugangsschutz durch das Betriebssystem... MS hat ein Kommandozeilen-Tool herausgegeben, mit dem die Sicherheitsstufe für Zertifikate und deren Privat-Keys erhöht werden kann. Genaueres dazu hat Steve Henson auf einer eigenen *eigenen Seite* `<http://www.drh-consultancy.demon.co.uk/cexport.html>` zusammengetragen.

CRLs bezieht der MSIE über eine im Zertifikat angegebene URI. Die URI wird über die Extension „CRL Distribution Points“ in ein Zertifikat gebracht. Leider unterstützt OpenSSL-0.9.2b diese Extension noch nicht. Eine Zertifikatprüfung erfolgt erst, wenn im Browser in der Rubrik „Ansicht / Internetoptionen / Erweitert“ im Abschnitt „Sicherheit“ das Kontrollkästchen „Auf zurückgezogene Zertifikate überprüfen“ angewählt ist.

5 Ergänzende Programme

5.1 `pfx-0.1.2` von Steve 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 MSIE ab Vers. 4.0x sollte statt `pfx` das `pkcs12`-Programm (siehe PKCS12 (5.2)) verwendet werden.

5.1.1 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 `SSLIB=$(SSLDIR)/lib` setzen.
- Zeile 3, `CFLAGS=-Wall -O2 -I$(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`

5.1.2 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 gelsden 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 (3.4)). 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 (5.1.2)).

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

5.2 `pkcs12-054` von Steve Henson

Das Programm arbeitet nur mit NSC ab Vers. 4.04 und MSIE ab Vers. 4.0 zusammen. Die Anwendung erfolgt analog der oben für `pfx` geschilderten Arbeitsweise. Mit dieser Applikation ist es - wie mit `pfx` für die „älteren“ Browser - möglich, aus einem Browser exportierte Zertifikate für OpenSSL zugänglich zu machen, sowie mit OpenSSL erzeugte Zertifikate in den Browser zu bringen. Das schließt den Import von CA-Zertifikaten über Zertifikatketten mit ein.

5.2.1 Übersetzung und Installation

Zur Übersetzung muß nach dem Auspacken der Programm-Quellen in das Verzeichnis `src` gewechselt werden. Folgende Änderungen sind dort im Makefile vorzunehmen:

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

- Zeile 3, `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 5, `SSLLIB=/usr/local/ssl/lib`:
  Pfad auf `SSLIB=$(SSLDIR)/lib` setzen.

- Zeile 7, `SSLBIN=/usr/local/ssl/bin`:
  Hier muß das Verzeichnis angegeben werden, in dem das pkcs12-Programm installiert werden soll, z.B. `SSLBIN=$(SSLDIR)/bin`

- Zeile 10, `TOP=/home/steve/ossl`:
  TOP muß auf das OpenSSL-Quellverzeichnis verweisen, also `TOP=$(SSLSRC)`

Schließlich ist das Programm mit

```
make clean ; make errors ; make 'CC=gcc'
```

zu übersetzen. Die Übersetzung erfolgt ohne Warnungen.

Als Installationsverzeichnis bietet sich `$(SSLDIR)` mit den entsprechenden Unterverzeichnissen an:

- `cp $(PKCS12_SRC)/src/pkcs12 $(SSLDIR)/bin`
- `chmod 755 $(SSLDIR)/bin/pkcs12`
- `cp $(PKCS12_SRC)/src/libpkcs12.a $(SSLDIR)/lib`
- `chmod 744 $(SSLDIR)/lib/libpkcs12.a`
- `cp $(PKCS12_SRC)/pkcs12.h $(SSLDIR)/include`
- `chmod 744 $(SSLDIR)/include/pkcs12.h`

5.2.2 Anwendung von pkcs12

Die Benutzung erfolgt ähnlich dem beim Programm `pfx` geschilderten Verfahren (siehe PFX (5.1.2)).

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.

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.

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
```

Eine Übersicht über alle Optionen findet sich in Anhang PKCS12 (D).

5.3 `ca-fix-0.3` von Steve Henson

Die von dem Programm CA-Fix bereitgestellte Funktionalität ist in OpenSSL-0.9.2b 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 und `[ User_CA ]` für die Herausgabe von Benutzer-Zertifikaten. Die 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.

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
pfad                = /usr/local/openssl


[ 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
#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


[ 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

# 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:1443/

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

nsComment               = This certificate was issued by a PCA

# 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.
# Zurueckgegeben wird dann 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
nsCertType              = server
nsBaseUrl               = https://mystic.pca.dfn.de:1443/
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?
# nsCaRevocationUrl     = cgi/CA-rev.cgi?
# nsRenewalUrl          = cgi/check-renw.cgi?


[ UCA_ext ]

# basicConstraints      = critical, CA:FALSE
keyUsage                = digitalSignature, keyEnciphermen, keyAgreement
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid,issuer:always
subjectAltName          = email:copy
issuerAltName           = issuer:copy
nsCertType              = client, email
nsBaseUrl               = https://mystic.pca.dfn.de:1443/
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?
# nsCaRevocationUrl     = cgi/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
```

```
nsBaseUrl               = https://mystic.pca.dfn.de:443/
nsCaPolicyUrl           = http://www.pca.dfn.de/dfnpca/policy/wwwpolicy.html
nsComment               = This certificate is a Root CA Certificate
nsRevocationUrl         = cgi/non-CA-rev.cgi?
# nsCaRevocationUrl     = cgi/CA-rev.cgi?
# nsRenewalUrl          = cgi/check-renw.cgi?

# 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

openssl

```
openssl

Standard commands
asn1parse      ca             ciphers        crl            crl2pkcs7
dgst           dh             dsa            dsaparam       enc
errstr         gendh          gendsa         genrsa         nseq
pkcs7          req            rsa            s_client       s_server
s_time         sess_id        speed          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-cbc        rc2-cfb
rc2-ecb        rc2-ofb        rc4            rc5            rc5-cbc
rc5-cfb        rc5-ecb        rc5-ofb
```

asn1parse

```
asn1parse [options] < infile

where options are
 -inform arg   input format - one of DER TXT PEM
 -in arg       input file
 -offset arg   offset into file
 -length arg   lenth of chapion 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
```

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
 -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
```

ciphers

```
usage: ciphers args

 -v          - verbose mode, a textual listing of the ciphers in SSLeay
 -ssl2       - SSL2 mode
 -ssl3       - SSL3 mode
```

crl

```
usage: crl args

 -inform arg     - input format - default PEM (one of DER, TXT 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
```

crl2pkcs7

```
crl2pkcs7 [options] < infile > outfile

where options are
 -inform arg    input format - one of DER TXT PEM
 -outform arg   output format - one of DER TXT 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'
```

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
```

dh

```
dh [options] < infile > outfile

where options are
 -inform arg   input format - one of DER TXT PEM
 -outform arg  output format - one of DER TXT 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
```

dsa

```
dsa [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
 -out arg      output file
 -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
```

dsaparam

```
dsaparam [options] [bits] < infile > outfile

where options are
 -inform arg   input format - one of DER TXT PEM
 -outform arg  output format - one of DER TXT PEM
 -in arg       input file
 -out arg      output file
 -text         check the DSA parameters
 -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
```

enc

```
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)
```

errstr

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

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
```

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
 -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
```

nseq

```
Usage nseq [options]

where options are
-in file  input file
-out file output file
-toseq    output NS Sequence file
```

pkcs7

```
pkcs7 [options] < infile > outfile

where options are
 -inform arg   input format - one of DER TXT PEM
 -outform arg  output format - one of DER TXT PEM
 -in arg       input file
 -out arg      output file
 -print_certs  print any certs or crl in the input
```

```
 -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
```

req

```
req [options] < infile > outfile

where options  are
 -inform arg    input format - one of DER TXT PEM
 -outform arg   output format - one of DER TXT 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.
 -asn1-kludge   Output the 'request' in a format that is wrong but some CA's
                have been reported as requiring
                [ It is now always turned on but can be turned off with -no-asn1-kludge ]
```

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
 -out arg      output file
 -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
```

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
-debug        - extra output
-nbio_test    - more ssl protocol testing
-state        - print the 'ssl' states
-nbio         - Run with non-blocking IO
-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       - prefered cipher to use, use the 'openssl ciphers'
                command to see what is available
```

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      - RSA 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)
-nbio         - Run with non-blocking IO
-nbio_test    - test with the non-blocking test bio
-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
 -bugs         - Turn on SSL bug compatability
 -www          - Respond to a 'GET /' with a status page
 -WWW          - Respond to a 'GET /<path> HTTP/1.0' with file ./<path>
```

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 compatability
-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       - prefered cipher to use, play with 'openssl ciphers'
```

sess_id

```
usage: sess_id args

 -inform arg     - input format - default PEM (one of DER, TXT 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
```

speed

```
speed

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
```

verify

```
usage: verify [-verbose] [-CApath path] [-CAfile fcert] cert1 1cert2 ...
```

version

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

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
 -serial         - print serial number value
 -hash           - print hash value
 -subject        - print subject DN
 -issuer         - print issuer DN
 -startdate      - notBefore field
 -enddate        - notAfter field
 -dates          - both Before and After dates
 -modulus        - print the RSA key modulus
 -fingerprint    - print the certificate fingerprint
 -noout          - no certificate output
 -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 asssumed 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 do an RSA sign with
```

C Optionen von pfx

Das Programm `pfx` von Steve Henson erlaubt es, Zertifikate und Private Keys in Netscape- und Microsoft-Browser zu importieren. *Hinweis:* Dieses Programm wurde durch das Programm `pkcs12` (s. Anhang PKCS12 (D)) ersetzt und wird hier nur der Vollständigkeit halber aufgeführt!

Import:

```
usage is pfx [options]

where: [options] is one or more of the following

-in file.p12     PFX file to read (default stdin).
-out file.pem    file to output to (default stdout).
-noout           don't output any certificates or keys just say if decryption
```

```
                 was OK.
-nokeys          don't output any private keys.
-print_certs     output all certificates.
-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.
```

Export:

```
usage is pfx -export -name NAME [options]

NAME is the certificate name (this appears in the listbox in Netscape) this
option is mandatory with -export .

-in file.pem     PEM file to read certificate from (default stdin).
-out file.p12    file to output to (default stdout).
-inkey file.pem  file to read private key from if not same as certificate.
-chain           add a complete chain of certificates (default just one).
-certfile c.pem  add all the certificates in c.pem
```

D Optionen von pkcs12

Das Programm pkcs12 von Steve Henson erlaubt es, u.a. Zertifikate und Private Keys als PKCS#12-Datei in Netscape- und Microsoft-Browser zu importieren.

```
ssleay pkcs12 Optionen
```

```
Usage: pkcs12 [options]

PKCS#12 program version 0.54
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)
-keyex        set MS key exchange type
-keysig       set MS key signature type
```

E 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* `<mailto:3weber@informatik.uni-hamburg.de>`).

F 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/>`)

F.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*

F.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/*
- ...