CSP Users Guide

Leif Johansson

Enheten för IT och Media Stockholms Universitet CSP version 0.20 (Mon May 28 16:20:54 CEST 2001)

1 Introduction

CSP is a perl wrapper around OpenSSL (*http://www.openssl.org*) which makes it easy to setup and run multiple certificate authorities with different configuration. CSP also includes support for generating a set of html files which can be used as a static certificate repository.

2 Obtaining and installing CSP

CSP is available for download from *http://devel.it.su.se/projects/CSP*. Apart from perl (version 5.6.1 or later) and OpenSSL (verson 0.9.6 or later) you need the perl-modules Date::Calc and Term::Prompt which you can find at your CPAN mirror (*http://www.cpan.org*). Install these prerequisites and then download the CSP tarball. Unpack CSP and install it using the following commands:

```
# zcat CSP-x.tar.gz | tar xvf -
...
# cd CSP-x
# perl Makefile.PL
# make
# make install
```

This will among other things install a script `csp` which is installed where perl installed its scripts (typically `/usr/local/bin` or `/usr/bin`). You should be able to run this script with the `--help` flag to get a summary of options:

```
# csp --help
Usage:  /usr/bin/csp <ca name> create

        /usr/bin/csp <ca name> delete

        /usr/bin/csp <ca name> init
              [--crtfile=<PEM certificate>]

        /usr/bin/csp <ca name> init
              [--keysize=<size>]
              [--keypass=<ca private key password>]
```

```
        [--keyfile=<private key file>]
        [--csrfile=<output PKCS10 request>]
        [--days=<ca certificate validity (days)>]
        [--email=<subjectAltName email>]
        [--url=<subjectAltName url>]
        [--digest=<sha1*|md5|md2|mdc2>]
        [--verbose]+
        <CA Subject (X509 Name)>

/usr/bin/csp <ca name> request
        [--keysize=<size>]
        [--keypass=<subject private key password>]
        [--keyfile=<private key file>]
        [--type=<*user|server|objsign|ca>]
        [--csrfile=<output pkcs10 request file>]
        [--noconfirm]
        [--verbose]+
        [--digest=<sha1*|md5|md2|mdc2>]
        {<X509 Name>|<RFC822 address>|<DNS name>}

/usr/bin/csp <ca name> issue
        [--keysize=<size>]
        [--keypass=<subject private key password>]
        [--keyfile=<private key file>]
        [--noconfirm]
        [--verbose]+
        [--type=<*user|server|objsign|ca>]
        [--days=<certificate validity (days)>]
        [--capass=<CA private key password>]
        [--email=<subjectAltName email>]
        [--url=<subjectAltName url>]
        [--ip=<subjectAltName ip address>]
        [--dns=<subjectAltName dns name>]
        [--digest=<sha1*|md5|md2|mdc2>]
        {<X509 Name>|<RFC822 address>|<DNS name>}

/usr/bin/csp <ca name> sign
        [--type=<*user|server|objsign|ca>]
        [--capass=<CA private key password>]
        [--csrfile=<input PKCS10 request>]
        [--email=<subjectAltName email>]
        [--url=<subjectAltName url>]
        [--ip=<subjectAltName ip address>]
        [--dns=<subjectAltName dns name>]
        [--digest=<sha1*|md5|md2|mdc2>]
        [--verbose]+
```

```
/usr/bin/csp <ca name> p12
       [--p12pass=<pkcs12 export password>]
       [--keypass=<private key password>]
       [--verbose]+
       <serial>

/usr/bin/csp <ca name> revoke <serial>
       [--noconfirm] [--quiet[=<level>]]

/usr/bin/csp <ca name> gencrl
       [--crldays=<days to next CRL update>]
       [--crlhours=<hours to next CRL update>]
       [--digest=<sha1*|md5|md2|mdc2>]
       [--verbose]+

/usr/bin/csp <ca name> genpublic
       [--export=<export directory>]
       [--verbose]+

/usr/bin/csp <ca name> list
       [--serial=<serial>]
       [--all]
       [--xinfo]
       [--contents]
       [--verbose]+

/usr/bin/csp --list

/usr/bin/csp --bundle
```

If you see this you are ready to configure CSP.

3 Configuring CSP

CSP should be run on a secure host. Ideally it should not be connected to a network. A good candidate is an old laptop dedicated for use as a CA host or a regular PC with a removable hard-drive which can be kept under lock and key when the CA is not in use. Before you can start to use CSP to create certifates you must

1. Configure CSP
2. Create a CA

If you decide to run CSP in a production environment on a separate host it is advisable to create a separate user which will login and issue certificates.

CSP depends on a set of configuration files to work. Examples of these files and a good starting-point for your setup can be found in the CSP tarball. Copy the directory `CSP-x/ca` to a directory where you will setup your CA. If you choose to run CSP as a separate user this directory will typically be this users homedirectory:

```
# pwd
/some/where/CSP-x
# su ca
> cd $HOME
> pwd
/home/ca
> cp -r /some/where/CSP-x/ca .
> chown -R ca ca
```

For the rest of the paper we will assume that the current working directory is the `$HOME` of this user or wherever the `ca` directory was copied. Now look inside this directory:

```
> tree ca
ca
|-- csp
`-- etc
    |-- aliases.txt
    |-- crl_extensions.conf
    |-- extensions.conf
    |-- oids.conf
    |-- public_html
    |   |-- certs
    |   |   |-- cert.html.mpp
    |   |   |-- index.html.mpp
    |   |   |-- revoked.html.mpp
    |   |   `-- valid.html.mpp
    |   `-- index.html.mpp
    `-- types.txt
```

The directory `csp` will eventually hold the certificate authorities you create. The directory `etc` contains configuration files. The files `aliases.txt`, `types.txt` and `oids.conf` are global configuration files but all other files are *templates* which are copied to each new CA as it is created. This means that if you want to make changes that affect all CA you must edit these files before creating production CAs.

A description of these files can be found in the appendix. For now it is enough to look at `extensions.conf` and `crl_extensions.conf`. These files are essentially parts of an openssl configuration file and share the same syntax. Take some time to take a look at these files before proceeding. Finally take a look at the files in the directory `public_html`. These files consititute a template for a public website of a CA. CSP does not contain a dynamic public interface (such as a CGI-script for searching the certificate database) but it does contain a means of generating and publishing such a public website. This website is static and is updated manually as a part of the regular maintenance which must be performed.

If you want to impose a common structure to the public websites of your CAs you should edit these files now. The format of these files is explained in the appendix.

Finally in order for csp to work you must set two environment variables. This is best done in your (or in the ca users) `.profile` or equivalent:

```
> CSPHOME=$HOME/ca
> OPENSSL=/where/your/openssl/lives/bin/openssl
> export CSPHOME OPENSSL
```

If you use csh/tcsh then use `setenv` instead:

```
> setenv CSPHOME $HOME/ca
> setenv OPENSSL /where/your/openssl/lives/bin/openssl
```

The `CSPHOME` variable must refer to the csp configuration directory and `OPENSSL` refers to the location of your `openssl` program. If you fail to set one or both of these variables csp will complain when you try to do anything beyond showing the list of options.

Even though csp is designed to manage multiple CAs it is only meant to be used by one user at a time. Running multiple copies of csp against the same `$CSPHOME` can have unexpected effects.

4 Creating a Certificate Authority (CA)

There are basically two ways to create a CA:

1. Create a self-signed ("root") CA
2. Create a CA subordinate to another CA

In both cases we begin the process by initializing the CA with the `csp create` command. This example shows the typical structure of all csp commands: The first argument is always the name of a CA and the second is always the csp command to execute on the CA.

```
> csp MyCA create
[CSP][MyCA    ] Successfully created CA MyCA
```

Here `MyCA` is the symbolic name of the newly created, but not yet operational certificate authority. This example also shows the way csp reports on progress. This command created a new directory named `MyCA` under `ca/csp`:

```
> tree ca/csp/MyCA
ca/csp/MyCA
|-- certs
|-- crl_extensions.conf
|-- extensions.conf
|-- index.txt
|-- private
|   `-- keys
|-- public_html
|   |-- certs
|   |   |-- cert.html.mpp
```

```
|   |   |-- index.html.mpp
|   |   |-- revoked.html.mpp
|   |   `-- valid.html.mpp
|   `-- index.html.mpp
|-- serial
`-- tmp
```

This listing is similar to to the above listing of `ca/etc` for good reason since most of the files above are simply copies of files from that directory. There are a few additional directories which will hold keys and certificates you issue, a file `serial` which contains the last serial number of a certificate issued from this CA and the usual OpenSSL certificate database (`index.txt`).

Now proceed with either of the next two sections depending on the type of certificate authority you are creating.

4.1 Creating a self-signed (root) CA

When creating a self-signed CA the contents of the CA certificate (extensions and standard fields) is determined by the configuration files in the newly created directory. That means that before completing the setup of this CA you must skip to the next section and at least edit `csp/csp/MyCA/extensions.conf` to reflect your situation.

Next proceed this way:

```
> csp MyCA init --keysize=2048 --days=365 \
  'CN=My Certificate Authority,O=Exempel AB,C=SE'
[CSP][MyCA    ] Generating CA key
[CSP][MyCA    ] Private key password:
[CSP][MyCA    ] Re-enter Private key password:
[CSP][MyCA    ] Successfully initialized self-signed CA MyCA
```

The options used would generate a CA valid for one year with a keysize of 2048 bits and with the distinguished name of `CN=My Certificate Authority,O=Exempel AB,C=S`. Typically all csp commands that issue requests or certificates takes as the final argument the distinguished name of the object to be produced. In this example the password for the CA private key was entered by hand at the prompt. Using the `--keypass` argument this can be sent to csp on the command-line.

4.2 Creating a subordinate CA

Setting up a subordinate request is a multi-stage process. First you must create a pkcs#10 request for a root certificate. Next send this request to the certificate authority which is going to sign the request and issue your root certificate. Next obtain your root certificate from the superior CA and import it into your CA making it operational. The first and last step is done using csp commands but the other steps (communicating with the superior CA) lies outside the scope of this guide.

First create the pkcs#10 request:

```
> csp MyCA init --keysize=2048 --csrfile=myca.csr \
  'CN=My Certificate Authority,O=Exempel AB,C=SE'
[CSP][MyCA    ] Successfully created CA MyCA
[CSP][MyCA    ] Generating CA key
[CSP][MyCA    ] Private key password:
[CSP][MyCA    ] Re-enter Private key password:
[CSP][MyCA    ] Successfully generated CA request for CA MyCA
```

Note the differences between this command and the one in the previous section where a self-signed certificate was created. The `--keysize` argument is the same and the distinguished name is present as the last argument but instead of a validity period a `--csrfile` argument is included. This argument refers to a file where the pkcs#10 file is to be created. Typically this file is then written to a diskette or sent in an email or posted in a web form to the superior CA.

Before sending the request to the superior CA take a moment to look it over using openssl:

```
> openssl req -text < myca.csr
Using configuration from /usr/local/ssl/openssl.cnf
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: CN=My Certificate Authority, O=Exempel AB, C=SE
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:db:5e:14:06:f8:39:cc:2a:b4:fa:ee:f4:18:48:
                    97:72:e1:54:28:16:cd:5b:72:68:2d:8e:bf:29:e2:
                    a0:54:7a:f0:0f:08:16:e7:25:94:fb:af:e0:e2:d4:
                    1c:a0:89:86:01:5b:15:ad:70:f3:1d:23:34:df:a8:
                    e2:ec:c0:ef:0d:06:4b:fa:03:9c:49:fb:a9:86:90:
                    da:c1:f1:a3:e6:9c:70:bd:f1:c2:9b:dd:d2:1c:12:
                    85:59:0c:dc:b1:dd:d4:e5:6c:09:60:e1:ad:b8:30:
                    d7:24:bd:c4:33:36:21:48:f6:d6:d0:e4:ca:94:8a:
                    ac:4e:47:94:b6:7a:17:23:6a:b5:fe:7c:88:69:99:
                    2a:d0:e1:84:7a:ae:f8:90:da:47:1a:13:44:43:41:
                    e0:76:50:de:92:b1:a0:9b:f1:3f:c7:fe:eb:59:8b:
                    e9:5f:55:7c:22:30:c4:3b:91:9f:56:a0:55:97:15:
                    ed:56:59:6a:5f:6d:78:94:2e:c1:d8:7d:56:51:f3:
                    fd:2a:e7:61:a2:1b:fe:8a:39:7d:be:50:1c:e1:38:
                    3c:03:23:f9:a3:b1:aa:1d:ca:a0:9c:f1:54:25:d0:
                    b6:1d:9f:62:6f:c2:76:07:f3:cb:4d:e2:4a:78:5c:
                    e4:58:3f:b1:ac:81:ed:13:e9:c9:bf:53:cf:e7:3f:
                    e4:d5
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
```

```
    Signature Algorithm: sha1WithRSAEncryption
        6a:d5:7a:23:89:84:ec:b1:e6:89:22:d3:72:08:f5:d4:33:16:
        2e:10:9e:7e:84:6b:a5:e2:55:6c:9e:5a:46:ef:43:14:1e:61:
        b4:c5:b5:10:07:a5:8e:5f:b5:65:de:6c:0f:69:42:45:3c:e3:
        b2:5b:ea:48:a3:27:69:ce:8c:da:b0:25:43:88:f9:1c:95:b9:
        22:e2:db:69:a4:19:5c:b3:74:d1:3d:ec:b6:7e:a8:2b:4f:2b:
        3d:f7:e6:ee:8f:bc:76:e4:83:a3:b6:a6:4b:d1:f2:f3:bc:35:
        cb:eb:23:45:cc:82:9b:2d:57:81:a8:a7:9f:be:2b:75:d2:6e:
        d0:a7:a4:78:4c:84:1f:4b:73:32:7c:7d:d1:25:c1:3a:d5:60:
        44:f4:2e:5b:e8:e3:ae:20:d2:8b:a8:6b:ef:0a:f5:bb:93:a4:
        73:58:84:e6:56:fb:c2:df:e7:8b:44:2a:96:31:69:8e:19:a8:
        07:a7:a9:6a:b1:c1:aa:16:f0:3b:93:e6:6a:94:5c:23:2c:7c:
        82:67:d9:e9:3d:86:80:68:cf:81:f5:ee:68:1d:93:d5:ca:73:
        b0:c9:cb:30:25:8a:7f:39:30:0d:32:dd:d5:7c:9e:c0:46:18:
        62:75:86:ba:a1:c4:06:84:2c:80:25:ac:4a:4d:e8:15:ec:3b:
        05:fa:1a:49
-----BEGIN CERTIFICATE REQUEST-----
MIICijCCAXICAQAwRTEhMB8GA1UEAxMYTXkgQ2VydGlmaWNhdGUgQXV0aG9yaXR5
MRMwEQYDVQQKEwpFeGVtcGVsIEFCMQswCQYDVQQGEwJTRTCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBANteFAb4OcwqtPru9BhIl3LhVCgWzVtyaC2Ovyni
oFR68A8IFucllPuv4OLUHKCJhgFbFa1w8x0jNN+o4uzA7w0GS/oDnEn7qYaQ2sHx
o+accL3xwpvd0hwShVkM3LHd1OVsCWDhrbgw1yS9xDM2IUj21tDkypSKrE5HlLZ6
FyNqtf58iGmZKtDhhHqu+JDaRxoTRENB4HZQ3pKxoJvxP8f+61mL6V9VfCIwxDuR
n1agVZcV7VZZal9teJQuwdh9VlHz/SrnYaIb/oo5fb5QHOE4PAMj+aOxqh3KoJzx
VCXQth2fYm/Cdgfzy03iSnhc5Fg/sayB7RPpyb9Tz+c/5NUCAwEAAaAAMA0GCSqG
SIb3DQEBBQUAA4IBAQBq1XojiYTsseaJItNyCPXUMxYuEJ5+hGul4lVsnlpG7OMU
HmG0xbUQB6WOX7Vl3mwPaUJFPOOyW+pIoydpzozasCVDiPkclbki4ttppBlcs3TR
Pey2fqgrTys99+buj7x25IOjtqZL0fLzvDXL6yNFzIKbLVeBqKefvit10m7Qp6R4
TIQfS3MyfH3RJcE61WBE9C5b6OOuINKLqGvvCvW7k6RzWITmVvvC3+eLRCqWMWmO
GagHp6lqscGqFvA7k+ZqlFwjLHyCZ9npPYaAaM+B9e5oHZPVynOwycswJYp/OTAN
Mt3VfJ7ARhhidYa6ocQGhCyAJaxKTegV7DsF+hpJ
-----END CERTIFICATE REQUEST-----
```

Note that by default csp always uses `sh1WithRSAEncryption` for all signatures. This algorithm is secure and should be a good choice for most applications.

Upon receipt of the ca certificate from the superior CA (stored in a PEM-encoded certificate file `myca.crt` in the current directory) do this:

```
> csp MyCA init --crtfile=myca.crt
[CSP][MyCA    ] Successfully initialized CA MyCA
```

This command finishes the setup of your subordinate CA. Read on to find out how to operate a certificate authority.

4.3 Post-configuration tasks

Before issuing certificates you must modify the configuration of your newly created CA. Take a look at the contents of the directory where `MyCA` is kept:

```
[leifj@njal pki-kurs]$ tree ca/csp/MyCA
ca/csp/MyCA
|-- ca.crt
|-- certs
|-- crl_extensions.conf
|-- extensions.conf
|-- index.txt
|-- private
|   |-- ca.key
|   `-- keys
|-- public_html
|   |-- certs
|   |   |-- cert.html.mpp
|   |   |-- index.html.mpp
|   |   |-- revoked.html.mpp
|   |   `-- valid.html.mpp
|   `-- index.html.mpp
|-- serial
`-- tmp
```

A few additional files have been created during setup. The important files are of course `ca.crt` and `private/ca.key` which is the ca certificate and the ca private key respectively.

First take a look at the main extensions section of `extensions.conf` (details about this file can be found in the appendix):

```
...

##
## These extensions are always present
##

nsCaRevocationUrl       = http://ca.example.com/crl-v1.crl
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid,issuer:always
authorityInfoAccess     = caIssuers;URI:http://ca.example.com/ca.crt
crlDistributionPoints   = URI:http://ca.example.com/crl-v2.crl
certificatePolicies     = ia5org,@certpolicy
issuerAltName           = email:ca@example.com,URI:http://ca.example.com
subjectAltName          = @altnames

[ altnames ]
```

```
%ifdef EMAIL
email.1                 = %{EMAIL}
%endif
%ifdef URI
URI.1                   = %{URL}
%endif
%ifdef DNS
DNS.1                   = %{DNS}
%endif
%ifdef IP
IP.1                    = %{IP}
%endif

...
```

You should edit this file to replace `ca.example.com` with the domain-name of your public CA website (to be covered later in this guide). Future versions of csp may do this automatically during setup of the CA. These extensions are always present in all certificates issued by this CA and consistitue a good set of defaults. Next look at the `certpolicy` section further down in the file:

```
[certpolicy]

policyIdentifier        = 1.1.1.1.1
## Map this to a real document in your webserver configuration
CPS.1                   = http://ca.example.com/CPS
userNotice.1            = @notice

[notice]

explicitText="Limited Liability, see http://ca.example.com/CP"
```

These sections must also be edited to reflect the OID of your certificate policy and the location of your certificate policy (CP) and certificate practice statement (CPS). The meaning of these terms are outside the scope of this document. If you are creating a CA without a policy (not recommended) you should remove these two sections and also the `certificatePolicies` attribute from the extensions section above.

Rembember that `ca/csp/MyCA/extensions.conf` affects certificates issued by this MyCA and not the contents of the CA certificate of MyCA itself. The content of this certificate is only created from this file in the case of a self-signed CA. If you are creating a self-signed CA you should edit this file before creating the CA certificate using `csp init` (see the section of self-signed CA above).

Normally this completes the setup of MyCA. If you need to add extensions to your certificate revocation lists (CRL) you should also take a look at `crl_extensions.conf`. However unless you know what you are doing you should refrain from changing that file. By default CSP produces both version 1 and version 2 CRLs and this file only affects the content of version 2 CRLs.

Finally you should modify the files containing the templates for the public website. These files are found in the directory `ca/csp/MyCA/public_html`. In principle you can have any template html-files in this directory

and when csp produces the public website (see below) any files found in this directory goes through a rewrite engine that takes all files ending in `.mpp` and produces a file with the same name without this extension. The only exception to this rule is the file `public_html/certs/cert.html.mpp` which is used as a template for each certificate in the database. More information about the structrure and syntax of these file can be found in the appendix.

At the very least you should edit these files and replace the default email adresses and contact information found there.

5 CA Operations

When your CA is operational you can start to issue certificates and performing other administrative tasks.

5.1 Issuing certificates

There are two ways to issue a certificate: either sign a pkcs#10 request you have obtained from a subject or issue a certificate directly without a request. The first case is most common in the case of a webserver or a subordinate CA. We will cover them in turn:

Assume we have received a request for a certificate for a webserver in the file `webserver.csr`. To sign it proceed as follows:

```
> csp MyCA sign --type=server --csrfile=webserver.csr
Using configuration from /usr/local/ssl/openssl.cnf
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: CN=www.example.com, O=Exempel AB, C=SE
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:c7:56:3c:7c:a2:a1:35:e7:67:4d:39:d3:97:85:
                    55:4e:71:d9:27:e8:c0:93:52:ce:47:11:49:a9:ff:
                    bb:1e:0f:8c:9e:fe:e2:a4:f2:d6:6a:20:62:dd:2c:
                    83:73:61:f8:1f:63:de:c0:33:32:06:f9:4d:ca:a8:
                    b2:3d:b9:78:c5:d1:e8:66:bb:f7:b0:4e:6c:c1:7d:
                    ca:54:c6:68:66:eb:ab:60:6f:c6:6f:89:e5:15:a5:
                    d9:3f:14:bd:8c:09:4c:78:52:21:09:31:68:9a:e4:
                    2f:d3:c2:83:c8:fc:2d:ac:da:2b:e9:5f:3c:fe:cc:
                    ff:bb:ec:b7:ca:2c:e3:6c:a9
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha1WithRSAEncryption
        33:5e:ae:7d:9a:fc:35:75:3f:24:f0:b2:2a:93:59:85:2c:80:
        fc:ee:4f:15:30:d3:2b:b2:a4:2b:01:52:77:e2:ec:16:32:e3:
```

```
        0f:40:85:89:1d:6a:a1:dc:14:4f:89:2c:fd:61:64:7a:93:59:
        33:9b:3d:bb:5f:42:18:81:91:b8:ec:68:3c:56:09:0b:a9:c6:
        da:d3:67:f5:bb:d4:85:be:1d:9d:9e:c2:07:0a:b0:4f:e8:9d:
        c5:e2:db:5e:33:fa:00:e3:a0:6b:2f:3a:9d:84:bd:d7:c3:89:
        3e:f1:b4:f3:f1:9f:9c:c9:e2:a6:67:6a:34:80:39:b6:29:35:
        64:3e
Really sign this? (y or n) [default n] y
[CSP][MyCA    ] Signing request
[CSP][MyCA    ] CA Private key password:
```

Remember that the password is not echoed on the tty (even as '*'s). Always look at the request before signing it. Next take a look at the contents of the certificate database:

```
> csp MyCA list
Serial  : 01
Status  : Valid
Subject : CN=www.example.com,O=Exempel AB,C=SE
Expires : Thu Jun 27 22:33:40 2002
```

The `--type=server` command-line argument is very important. It tells csp which set of extensions to select in `ca/csp/MyCA/extensions.conf`. Look at the top of this file again:

```
...

##
## Extensions by certificate type
##

%ifdef TYPE_CA
basicConstraints        = critical,CA:TRUE
keyUsage                = critical,cRLSign, keyCertSign
nsCertType              = sslCA, emailCA, objCA
%endif

%ifdef TYPE_USER
nsCertType              = client, email
keyUsage                = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage        = clientAuth,emailProtection
%endif

%ifdef TYPE_OBJSIGN
nsCertType              = objsign
keyUsage                = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage        = codeSigning
%endif

%ifdef TYPE_SERVER
```

```
nsCertType              = server
keyUsage                = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage        = serverAuth
%endif

...
```

Since we said `--type=server` in our signing command the last section of this file was used when producing the certificate. This is significant since it allows this certificate to be used as an SSL server certificate and also causes the certificate to contain the right netscape extension for server use. Unless specified the type defaults to `user` which is something else altogether.

You could have issued the same certificate directly by the following command:

```
> csp MyCA issue --type=server --days=365 'CN=www.example.com,O=Exempel AB,C=SE'
[CSP][MyCA    ] Generating new key
[CSP][MyCA    ] Private key password:
[CSP][MyCA    ] Re-enter Private key password:
[CSP][MyCA    ] Create certificate request for CN=www.example.com,O=Exempel AB,C=SE
Using configuration from /usr/local/ssl/openssl.cnf
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: CN=www.example.com, O=Exempel AB, C=SE
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:c3:6f:20:bd:50:4b:a2:4e:16:3d:b7:d2:b4:56:
                    9f:35:90:10:f8:4f:74:83:0b:2c:90:bb:25:a0:cc:
                    22:8f:e5:d4:53:ff:60:36:a2:b3:9d:21:a4:5d:67:
                    bb:6f:06:a0:e9:6a:66:dc:ce:54:16:72:43:fc:be:
                    e7:a1:a3:d7:21:1e:ff:6a:53:b2:1b:d0:1c:b3:e0:
                    85:1f:bd:ab:b0:a9:21:bd:8f:da:80:44:e6:98:a4:
                    a3:9c:01:fe:38:54:1c:5e:a6:bb:a2:81:a6:a8:46:
                    3d:91:f4:17:2c:35:99:f5:0e:00:fd:2c:84:e9:a1:
                    d4:d0:c8:73:82:00:cd:4e:39
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha1WithRSAEncryption
        03:e0:90:66:9a:bf:0d:24:25:4c:c8:a8:4a:2d:29:f7:fa:3c:
        45:06:0a:e3:76:41:de:fc:d7:08:b1:0a:3c:a6:38:98:41:6a:
        64:d9:58:f9:e2:55:ea:65:ad:e5:6c:b9:e3:48:21:dc:85:44:
        a8:ac:29:af:d3:86:ee:c0:d9:4d:5c:59:f0:b9:3a:1d:fa:76:
        39:48:be:cb:61:de:e1:a0:a9:88:86:52:ef:d3:3a:d1:5a:72:
        2e:6c:e3:3e:d8:cc:4e:fa:6d:fa:2c:43:ba:6b:c5:95:57:a4:
        1c:7b:da:6f:38:a2:ff:f1:9e:d0:5c:f5:59:e8:24:d3:74:b6:
```

```
        80:61
Really sign this? (y or n) [default n] y
[CSP][MyCA    ] Signing request
[CSP][MyCA    ] CA Private key password:
```

The astute reader will note that what is really happening here is that a request is first created and then signed. The above command is actually equivalent to the following commands (abbreviated output):

```
> csp MyCA request --csrfile=tmp.csr 'CN=www.example.com,O=Exempel AB,C=SE'
...
> csp MyCA sign --type=server --days=365  --csrfile=tmp.csr
...
```

5.2 Revoking certificates

Assume we need to revoke the newly created certificate:

```
> csp MyCA revoke 01
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: CN=My Certificate Authority, O=Exempel AB, C=SE
        Validity
            Not Before: May 27 22:33:40 2001 GMT
            Not After : May 27 22:33:40 2002 GMT
        Subject: CN=www.example.com, O=Exempel AB, C=SE
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:c7:56:3c:7c:a2:a1:35:e7:67:4d:39:d3:97:85:
                    55:4e:71:d9:27:e8:c0:93:52:ce:47:11:49:a9:ff:
                    bb:1e:0f:8c:9e:fe:e2:a4:f2:d6:6a:20:62:dd:2c:
                    83:73:61:f8:1f:63:de:c0:33:32:06:f9:4d:ca:a8:
                    b2:3d:b9:78:c5:d1:e8:66:bb:f7:b0:4e:6c:c1:7d:
                    ca:54:c6:68:66:eb:ab:60:6f:c6:6f:89:e5:15:a5:
                    d9:3f:14:bd:8c:09:4c:78:52:21:09:31:68:9a:e4:
                    2f:d3:c2:83:c8:fc:2d:ac:da:2b:e9:5f:3c:fe:cc:
                    ff:bb:ec:b7:ca:2c:e3:6c:a9
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            Netscape Cert Type:
                SSL Client, S/MIME
            X509v3 Key Usage:
```

```
            Digital Signature, Non Repudiation, Key Encipherment
        X509v3 Extended Key Usage:
            TLS Web Client Authentication, E-mail Protection
        Netscape CA Revocation Url:
            http://ca.example.com/crl-v1.crl
        X509v3 Subject Key Identifier:
            93:E9:F4:2F:1C:6C:0B:90:D9:DC:5E:59:15:FA:A5:46:5E:77:52:D1
        X509v3 Authority Key Identifier:
            keyid:07:40:9A:AA:59:C8:36:8B:EA:74:80:21:39:3B:2D:FE:87:47:38:C4
            DirName:/CN=Root CA/O=Exempel AB/C=SE
            serial:02

        Authority Information Access:
            CA Issuers - URI:http://ca.example.com/ca.crt

        X509v3 CRL Distribution Points:
            URI:http://ca.example.com/crl-v2.crl

        X509v3 Certificate Policies:
            Policy: 1.1.1.1.1
              CPS: http://ca.example.com/CPS
              User Notice:
                Explicit Text: Limited Liability, see http://ca.example.com/CP

    Signature Algorithm: sha1WithRSAEncryption
        47:b7:30:4a:b9:c5:9e:76:a5:9b:d9:c4:96:b6:75:fe:4f:a0:
        d6:84:db:29:73:47:ed:23:13:c0:ae:90:33:de:28:18:71:d5:
        de:28:fe:33:8a:0c:56:e0:dd:a1:95:ae:0d:a4:9a:fc:12:e8:
        07:65:9d:10:71:45:f2:b2:79:66:f6:9b:fc:1d:10:39:8c:6c:
        da:b1:4f:98:6b:16:61:f2:89:5d:35:7d:54:2a:8b:29:8b:0d:
        e4:52:1c:71:8e:39:0c:22:00:86:2d:44:5b:16:dc:74:48:f0:
        16:d4:09:22:3a:3d:13:a4:c3:4e:30:6d:d0:58:97:05:dc:12:
        f1:23:fb:40:30:0c:28:e7:c2:ac:b4:d4:bc:d1:89:a5:b1:bc:
        a4:20:64:25:f7:70:a3:b4:4e:b5:23:ab:c7:b4:d1:16:35:58:
        a9:ec:23:f1:35:88:1e:7d:51:be:40:13:7f:13:b7:b4:7a:f3:
        ba:21:83:b4:a1:26:f5:71:d3:4e:21:89:89:14:3e:64:9d:93:
        02:86:44:d8:89:3b:47:c6:64:08:4b:f6:ba:ce:58:9d:60:6e:
        f9:a6:d2:cd:ee:35:dd:a7:1b:0c:34:40:8d:3c:0a:a8:b5:2a:
        e9:b5:a2:cb:38:22:69:5a:da:4f:70:cb:2b:f3:22:9c:1a:b1:
        1e:1d:d2:92
Really revoke this? (y or n) [default n] y
[CSP][MyCA    ] CA Private key password:
```

Note that the last argument in this case (the serial number of the certificate) must be entered litterally, i.e `01` and not `1`

Next list the contents of the certificate database again, this time using the `--all` switch to include both revoked and valid certificates:

```
> csp MyCA list --all
Serial  : 01
Status  : Revoked
Subject : CN=www.example.com,O=Exempel AB,C=SE
Expires : Thu Jun 27 22:33:40 2002
Revoked : Wed Jun 27 22:43:31 2001
```

Revoking a certificate is not enough though. You must also create and publish a new certificate revocation list (CRL).

5.3 Exporting certificates and private keys

In a perfect world you should never generate private keys on the certificate server and only sign pkcs#10 request. In practice however it is often necessary to issue certificate based on a non- cryptographically signed request (such as a verbal or written request). In this case the private key is created by csp and stored in the directory `private/keys` relative to the CA directory.

To export the private key in this format simply transfer it to removable media. Note that csp does not require the key to be present in the database and the key can be removed when the subject has received her copy. Also note that the key is protected with the password chosen when the certificate was issued.

A more secure way to export a combination of a certificate and a private key is to create a pkcs#12 object. This cryptographically protected structure combines a certificate chain and a private key in a single file which is protected by a password (sparate from the private key password) and which can be transported in the open to the subject. Such a structure can be imported into many browsers to be used as a user certificate. Alternatively it can be unpacked using openssl (for instance) to a raw private key and certificate to be used in a webserver.

To produce a pkcs#12 object of an issued certificate/private keypair use the following command:

```
> csp MyCA p12 01
[CSP][MyCA    ] Private key password:
[CSP][MyCA    ] PKCS12 export password:
```

Here `01` is the serial of the certificate you wish to export. If this certificate or the corresponding private key is not found in the database csp is unable to produce the pkcs#12 object.

The pkcs#12 objects are stored in the directory `p12` relative to the CA directory:

```
> tree ca/csp/MyCA/p12
ca/csp/MyCA/p12
‘-- 02.p12
```

5.4 Perioding maintenance

To successfully operate your CA you must perform certain tasks at regular intervals such as generating and publishing certificate revocation lists and updating the public website. Note that the public website is the prefered way to export newly created certificates. This means that when you issue certificates and when you generate a new CRL you should always generate and publish a new public website.

5.4.1 Generating CRLs

To generate a certificate revocation list do the following:

```
> csp MyCA gencrl
[CSP][MyCA   ] CA Private key password:
```

This produces both a version 1 and version 2 crl in `ca/csp/MyCA`. How often CRLs should be created is typically specified in the certificate policy or certificate practice statement for your CA. Each CRL contains a field which says when the next update is due. By default csp annonces a new update in a CRL 30 days in the future. This means that even if you never issue (or revoke) a single certificate your CRLs must be updated every 30 days. Using the `--crldays` and `--crlhours` switch you can change the time between CRL updates.

It is usually a good idea to publish a new CRL each time a certificate is revoked so that new users is informed of the revocation immediately.

5.4.2 Producing the public website

Each time you update your CRLs or publish a new certificate you must create and publish a new public website. Since csp is designed to be run on a secure and isolated host the typical scenario is that the public website is produces on the CA host and transported to the public website using a read-only medium. Here is how to produce the public website:

```
> csp MyCA genpublic --export=/dev/fd0
> tree /dev/fd0
/dev/fd0
|-- ca.crt
|-- certs
|   |-- 01.crt
|   |-- 01.html
|   |-- 01.pem
|   |-- 02.crt
|   |-- 02.p12
|   |-- 02.html
|   |-- 02.pem
|   |-- index.html
|   |-- revoked.html
|   `-- valid.html
|-- crl-v1.crl
|-- crl-v2.crl
`-- index.html
```

In this example the CA has been in operation and has issued two certificates one of which has been subsequently revoked. The export directory (in this case a floppy) does not contain any sensitive information and can be publised as is. One way is to setup a (virtual) webserver somewhere and copy the entire contents of the export directory to that webservers `htdocs` directory.

Note that a pkcs12-object is present in the certs directory. Since pkcs12 objects contain (twice encrypted) private keys it may be a good idea to remove these files from the certificate database when the user has been given time to download the file. That way when the public website is generated the pkcs12 object will not be included or refered to in the html-files.

When updating the public website make sure to delete any contents not generated by csp before copying the contents of the export medium to the public webserver directory.

A CSP Architecture

One of the design goals behind csp was to be able to support types of distinguished names not easily handled by static OpenSSL configuration files notably DC-style names. Although not mentioned in detail above the problem with DC-style names in OpenSSL is that the hierarchy is (for the purpouse of this excercise) arbitrarily deep and at each level the same attribute is used.

To solve this problem csp assembles a temporary openssl configuration file each time such a file is needed by any of the `openssl` commands. This structure allows the configueration to be changed at each invocation of csp withouth the need for editing configuration files simply by allowing command-line arguments to affect the contents of the temporary configuration file.

B Debugging CSP

To see the output from openssl commands executed by csp simply include the switch `--verbose` (which can be included multiple times for more output). Conversely csp can be less chatty by including the switch `--noconfirm` which causes csp to stop asking permission to do various things (like signing requests).

An important tool in debugging csp can be the temporary configuration files. These are normally deleted on completion of each openssl command but if you define the environment variable `CSPDEBUG` these files are saved in `tmp` relative to the CA directory. Setting this variable also causes csp to echo each openssl command as it is run.

C Known bugs

Due to a bug in OpenSSL csp sometimes says "using configuration from /usr/local/ssl/openssl.conf" or wherever you have openssl installed. This message can be ignored.

There are undoubtedly many other bugs waiting to be discovered. Report them to `csp-bugs@it.su.se`.

D CSP Configuration file syntax

The main CSP configuration files `extensions.conf` and `crl_extensions.conf` are essentially partial OpenSSL configuration files and share syntax with that file. Apart from standard OpenSSL syntax describing extensions csp adds a simple macro language which can be used to change the contents of the configuration file based on the csp command and the command-line arguments. This language is based (loosly) on Transarc mpp (macro pre processor) and contain the following set of constructions:

1. `%if`/`%endif`
2. `%ifdef`/`%endif`
3. `%include`
4. `%{`*variable*`}` (variable expansion)

The most commonly used is `%ifdef` which takes as its first argument a variable. The enclosing block is included in the output if and only if the variable is "defined". The following is a list of variables available when issuing certificates or signing pkcs#10 requests:

1. `TYPE_`*type*: The type of certificate beeing created.
2. URI, EMAIL, DNS and IP: These correspond to the command-line arguments to `csp issue` and `csp sign` and contain the various forms of `subjectAltName`.

When generating the certificate html files (`certs/`*serial*`.html` from `certs/cert.html.mpp`) in the public website the following variables are available:

1. `DATE` printable version of the current date
2. `SUBJECT_SERIAL` the certificate serial number
3. `SUBJECT_DN` the distinguished name of the subject
4. `ISSUER_DN` the distinguished name of the issuer
5. `SUBJECT_SHA1` the SHA1 fingerprint of the certificate
6. `SUBJECT_MD5` the MD5 fingerprint of the certificate
7. `SUBJECT_NOTBEFORE` and `SUBJECT_NOTAFTER` the validity period of the certificate (printable dates)
8. `SUBJECT_PKCS12` this is defined when a pkcs12 object exists for the certificate/keypair.

Finally when generating all other html files in the public website the following variables are available:

1. `DATE` printable version of the current date
2. `VALID` an html table with links to valid certificates
3. `VALID_COUNT` the number of valid certificates
4. `REVOKED` an html table with links to revoked certificates
5. `REVOKED_COUNT` the number of revoked certificates
6. `EXPIRED` an html table with links to expired certificates
7. `EXPIRED_COUNT` the number of expired certificates
8. `SUBJECT_SERIAL` the CA certificate serial number
9. `SUBJECT_NOTBEFORE` and `SUBJECT_NOTAFTER` the validity period of the CA certificate (printable dates)
10. `SUBJECT_DN` the distinguished name of the CA certificate
11. `SUBJECT_SHA1` the SHA1 fingerprint of the CA certificate
12. `SUBJECT_MD5` the MD5 fingerprint of the CA certificate

E CSP Configuration file reference

E.1 extensions.conf

```
##
## This is a prototype extension specification file
## which is processed using a macro language similar
## to transarc mpp. Do not delete the "[ extensions ]"
## section header.
##

[ extensions ]

##
## Extensions by certificate type
##

%ifdef TYPE_CA
basicConstraints        = critical,CA:TRUE
keyUsage                = critical,cRLSign, keyCertSign
nsCertType              = sslCA, emailCA, objCA
%endif

%ifdef TYPE_USER
nsCertType              = client, email
keyUsage                = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage        = clientAuth,emailProtection
%endif

%ifdef TYPE_OBJSIGN
nsCertType              = objsign
keyUsage                = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage        = codeSigning
%endif

%ifdef TYPE_SERVER
nsCertType              = server
keyUsage                = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage        = serverAuth
%endif

##
## These extensions are always present
##

nsCaRevocationUrl       = http://ca.example.com/crl-v1.crl
subjectKeyIdentifier    = hash
```

```
authorityKeyIdentifier  = keyid,issuer:always
authorityInfoAccess     = caIssuers;URI:http://ca.example.com/ca.crt
crlDistributionPoints   = URI:http://ca.example.com/crl-v2.crl
certificatePolicies     = ia5org,@certpolicy
issuerAltName           = email:ca@example.com,URI:http://ca.example.com
subjectAltName          = @altnames

[ altnames ]

%ifdef EMAIL
email.1                 = %{EMAIL}
%endif
%ifdef URI
URI.1                   = %{URL}
%endif
%ifdef DNS
DNS.1                   = %{DNS}
%endif
%ifdef IP
IP.1                    = %{IP}
%endif

[certpolicy]

policyIdentifier        = 1.1.1.1.1
## Map this to a real document in your webserver configuration
CPS.1                   = http://ca.example.com/CPS
userNotice.1            = @notice

[notice]

explicitText="Limited Liability, see http://ca.example.com/CP"
```

E.2 crl_extensions.conf

```
##
## This is a prototype CRL extension specification file
## which is processed using a macro language similar
## to transarc mpp. Do not delete the "[ crl_extensions ]"
## section header.
##

[ crl_extensions ]

authorityKeyIdentifier  = keyid,issuer:always
```