Introduction to the openssl command tool

The openssl command tool supports just under a hundred commands. Each of these can be configured using a variety of different flags. It is therefore beyond the scope of this worksheet to discuss them all.

The commands that we are most likely going to use during this assignment are:

- `ca`
- `genrsa`
- `req`
- `x509`
- `pkcs12`

Note: If we just type openssl on the command line you get to the interactive mode. Although there is no help if you type help it moans and displays all the possible commands!

The following section covers the most likely functions we'll need to use.

Generating a key

To generate an RSA key:

```
# openssl genrsa -out privkey.pem 1024
```

This creates a 1024 bit RSA key.

Commands and flags:

- **`genrsa`** – Generating RSA keys
- **`-out`** - Specify where to store the key.
- **`1024`** – Size of the key in bits

Note: If we omit the `-out` flag the output will go to the stdout.

Generating a certificate request

In order to create a certificate we first need to generate a certificate request. We can then send a CA the request and they will issue a signed certificate.

A command like:

```
# openssl req -newkey rsa:1024 -sha1 -keyout mykey.pem -out myreq.pem
```

… generates a certificate request. During this process a private key is created and stored in the mykey.pem file. The certificate request is stored in the myreq.pem file. Another common format for certificate requests is the csr format (certificate signing request).

Commands and flags:

- **`req`** – So as the name suggests the command is used for making certificate requests
- **`-keyout -`** Specify where to store the private key.
- **`-out`** - Specify where to store the request.
- **`-sha1`** – Message digest flag.
- **`-newkey rsa:1024`** – Generate a new 1024 bit RSA key.

Viewing a certificate request

If we want to look at the certificate request that was just created:

```
$ openssl req –noout –text –verify –in myreq.pem
```

Commands and flags:

- **`-noout`** – To the stdout (don't encode).
- **`-verify -`** Check the signature.

Issuing a certificate

Once we have certificate request we need to get a CA to issue us with a certificate.

```
# openssl x509 –req –in myreq.pem –sha1 –signkey mykey.pem –out
mycert.pem
```

This command will issue the mycert.pem certificate. Whilst running this command we will be asked to verify that we wish to sign and issue the certificate.

Commands and flags:

- **`-in`** - So as the name suggests the command is used for making certificate requests
- **`signkey -`** Specify where to store the private key.
- **`-out`** - Specify where to store the request.
- **`X509`** – This command indicates we are working with x509 files.

Viewing a certificate

To view a certificate :

```
$ openssl x509 –in mycert.pem –text –noout
```

This will display information about the certificate such as the public key, serial number, details about the issuer etc.

Commands and flags:

- **`-text`** – Human readable.

- **-noout** – To the stdout.

Creating DH keys

The command for creating a dh key is:

```
# openssl dhparam –check –text –5 512 –out dh512.pem
```

Commands and flags:

- **dhparam** – DH parameter manipulation and generation.
- **-5** – The generator to be used (can also be -2).

Creating a CA

This method illustrates creating a CA without the use of configuration files such as openssl.cnf. The following command creates the self-signed certificate and private key:

```
$ openssl req –x509 –newkey rsa:1024 –keyout myCAPrivKey.pem –days 14
–out myCAcert.pem
```

Enter details as requested. Remember if we used a configuration file it could be read from there instead.

Commands and flags:

- **-x509** – Self signed x509.
- **-days** 14 – The certificate will be valid for 14 days.

Now it is possible to generate requests for and issue certificates. So in the same way as before we can generate the certificate request:

```
$ openssl req –new –keyout myPrivKey.pem –out myCertReq.pem
```

Commands and flags:

- **-new** – This is a new request so ask all the DN questions

Now the CA created in step 1 is used to issue the certificate based upon the request just created:

```
$ openssl x509 –req –in myCertReq.pem –CA myCAcert.pem –CAkey
myCAPrivKey.pem –CAcreateserial –out ksb_cert.pem –days 14
```

The import things here are the –CA and the –CAkey flags. They are used to indicate the location of the certificate authority's certificate and private key. Certificates need serial numbers, the –CAcreateserial flag means create the serial file if it doesn't exist. (It will be named after the CA's certificate.)

Commands and flags:

- **-CA** – The certificate authorities certificate..
- **-CAkey** – The certificate authorities private key.
- **-CAcreateserial** – Create the serial file for the CA.
- **-days** 14 – The certificate will be valid for 14 days.

Converting from PEM to pkcs12

We may find at some point that we require a certificate to be in a form other than PEM so the following command illustrates how to create a certificate in pkcs12 format using a PEM certificate.

```
$ openssl pkcs12 -export -in mycert.pem -inkey myPrivkey.pem -out
mypkcs12cert.p12 -name "My PKCS12 Cert"
```

This command converts the certificate specified by the –in flag and the private key specified by the –inkey flag and creates the mypkcs12cert.p12 certificate (specified by the –out flag).

Commands and flags:

- **pkcs12** – Indicates we are using the pkcs12 format for working with files
- **-export -** This flag indicates we are exporting from the PEM files to the .p12 file.
- **-inkey** - Specify where to find the private key.
- **-name** – The name! (Friendly name)

Viewing a pkcs12 certificate file

If we want to view the contents of the pkcs12 certificate we have just created:

```
$ openssl pkcs12 -info -in mypkcs12cert.p12
```

Commands and flags:

- **-info** – Display the information held in the file

Creating a CA environment

Have a look at where openssl is installed, for the purposes of this document we'll refer to this as the openssl home directory. This will probably be at one of these locations but may change depending on the configuration chosen by our network administrator.

```
/usr/local/ssl/openssl.cnf
/usr/ssl/openssl.cnf
/usr/local/openssl/openssl.cnf
/usr/openssl/openssl.cnf
```

Inside this directory we will find 4 folders and one configuration file. The files we are interested in are the certs and private directories and the openssl.cnf. The certs

directory holds certificates that this CA has issued. The private directory holds the private key for the CA. And finally the openssl.cnf holds the configuration data for the CA. This file is looked at in more detail in the next section.

So to set up our environment we need create the following directories and files:

Note: For this example I'm setting my CA home directory as: /home/ian/iansCA

Create the iansCA directory
Inside this directory create the certs and private directories
Create a file called serial and store the number 01 in it.
Create an empty file called index.txt

Now you just need to create a CA configuration script!

Notes and Pointers

Windows: If you are more comfortable under a Windows environment then you might like to try Cygwin (cygwin.com). Download this and when you run it for the first time either select all packages and leave it running for a good hour or select most of the development and the openssl package.

Certificates: Creating certificates can be a cumbersome task and no doubt you won't get it right first time so why not create your certificates inside your makefile.

Web Browsers: What format does a web browser expect a certificate to be in?

Passwords: Pick one password and use it for all certificates, authorities, keys etc

Configuration files: Why not create a configuration file for each certificate and CA then automate the building of your application in a makefile.

Questions

These are an example of the types of question you should be asking yourself:

- What is a public key and what is a private key?
- What is PEM?
- What is the relationship between certificates and keys?
- Which comes first the certificate or the key?
- What is the serial file for?
- What type of certificates do browsers typically require?

Glossary of Terms

Try and create a glossary of all the terms you encounter. Here are a few to start you off:

PEM
X509
RSA
SHA
CA
DH

Appendix 1: Sample configuration file

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

# This definition stops the following lines choking if HOME isn't
# defined.
HOME            = .
RANDFILE        = $ENV::HOME/.rnd

# Extra OBJECT IDENTIFIER info:
#oid_file       = $ENV::HOME/.oid
oid_section     = new_oids

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

[ 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  = CA_default            # The default ca section

####################################################################
[ CA_default ]

dir         = ./demoCA        # 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.
#unique_subject   = no              # Set to 'no' to allow creation
of
                                # several ctificates with same subject.
new_certs_dir     = $dir/newcerts         # default place for new
certs.

certificate = $dir/cacert.pem       # The CA certificate
serial            = $dir/serial           # The current serial number
#crlnumber  = $dir/crlnumber  # the current crl number
                                # must be commented out to leave a V1
CRL
crl         = $dir/crl.pem          # The current CRL
private_key = $dir/private/cakey.pem# The private key
RANDFILE    = $dir/private/.rand    # private random number file

x509_extensions   = usr_cert        # The extentions to add to the
cert
```

```
# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt    = ca_default            # Subject Name options
cert_opt    = ca_default            # Certificate field options

# Extension copying option: use with caution.
# copy_extensions = copy

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

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

# 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

# For the CA policy
[ policy_match ]
countryName       = match
stateOrProvinceName     = match
organizationName  = match
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       = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName  = optional
organizationalUnitName  = optional
commonName        = supplied
emailAddress            = optional

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

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

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

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

[ req_distinguished_name ]
countryName             = Country Name (2 letter code)
countryName_default             = AU
countryName_min                 = 2
countryName_max                 = 2

stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = Some-State

localityName                    = Locality Name (eg, city)

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = Internet Widgits Pty Ltd

# 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     =

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

emailAddress                    = Email Address
emailAddress_max        = 64

# SET-ex3               = SET extension number 3

[ req_attributes ]
challengePassword       = A challenge password
challengePassword_min           = 4
challengePassword_max           = 20

unstructuredName        = An optional company name

[ usr_cert ]

# These extensions are added when 'ca' signs a request.

# 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=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType                    = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment                = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl             = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

[ v3_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]


# Extensions for a typical CA


# PKIX recommendation.
```

```
subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer:always

# This is what PKIX recommends but some broken software chokes on
critical
# extensions.
#basicConstraints = critical,CA:true
# So we do this instead.
basicConstraints = CA:true

# Key usage: this is typical for a CA certificate. However since it
will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
# keyUsage = cRLSign, keyCertSign

# Some might want this also
# nsCertType = sslCA, emailCA

# Include email address in subject alt name: another PKIX
recommendation
# subjectAltName=email:copy
# Copy issuer details
# issuerAltName=issuer:copy

# DER hex encoding of an extension: beware experts only!
# obj=DER:02:03
# Where 'obj' is a standard or added object
# You can even override a supported extension:
# basicConstraints= critical, DER: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
```