

OpenSSL

Introduction

OpenSSL is a free implementation of Secure Socket Layers, allowing you to create, manage, and sign SSL certificates. OpenSSL has created executables, static and dynamic libraries for use on all platforms, making it an extremely useful tool. The documentation below describes the steps from creating your own Certificate Authority to signing server and personal certificates.

You can also use this CA to sign developer certificates (a.k.a. object-signing certificates), but it is recommend you use a commercial CA since they have their public keys installed with most browsers and do not require the developers to ensure that their company's self-signing CA is installed on all possible clients for the signed code. Information on how to sign objects, therefore, is contained within a separate document, but follows the instructions below as far as signing the certificate with the CA that will be created here.

Index

OpenSSL	1
Introduction	1
Index	1
Definitions	2
Requirements	2
Creating a Self-signing CA	3
Description	3
Creating the CA	3
Creating a Server Certificate	4
Description	4
Creating the Certificate Request	4
Signing Your Certificate with You CA	5
Description	5
Signing Your Certificate	6
Creating a Personal Email Certificate	6
Description	6
Creating Your Personal Email Certificate	6
Exporting Your Personal Email Certificate	7
Description	7
Packaging Your Keys for Importing Later	7
Installing the Certificates	8
Description	8
Installing Self-Signing CA's	8
Server Certificates	12
Personal Email Certificates	12
Summary	18
Appendix A: Configuration	19
Acknowledgements	24

Definitions

- CA Certificate Authority. A self-signing certificate that allows an organization to sign other certificates such as server and personal certificates for email.
- Certificate An SSL key that describes who or what you are with a public key used to verify against the server's private key. Establishes secure and trusted communications.
- Private Key A secure key (typically using the RSA algorithm) for use in decrypting data and signing your data to be verified by clients. You must keep secure so that others cannot download or otherwise obtain.
- Public Key A public key that you distribute to clients for use in encrypting data such as HTTP connections or email messages, and verifying signed data.
- SSL Secure Sockets Layer. Current, the standard is version 3, a.k.a. SSLv3.

Requirements

- OpenSSL The libraries and executables to implement SSL. These tools are available from <http://www.openssl.org> for both Linux and Win32. Macintosh ports are also in the works. There is currently a ZIP file of containing debug and release builds for both static and dynamic libraries of OpenSSL, as well as the main executable in [\COLUMBUS\Projects](#). The file name is openssl.zip. Make sure the openssl.exe file is in your PATH environment variable.
- OS Operating system requirements are different depending on the platform for which you downloaded OpenSSL. Builds currently include support for both Unix/Linux/BSD and Win32.
- Browser You will need a browser to test this functionality. I recommend Internet Explorer 4.0 or above as it is the easiest. Netscape 4.0 or above will also work. Keep in mind, however, that almost any browser can use SSL, the two above just make it easy to implement.
- Configuration You need a configuration file for this to work. Create a directory /usr/local/ssl (for *nix) or \usr\local\ssl (for win32) and copy the text from [Appendix A: Configuration](#) to a file in the above directory called "openssl.cnf". If you do not do this, openssl will crash.
- Working Dir .. You need to have a working directory. I would recommend the directory you created above (/usr/local/ssl or \usr\local\ssl) since your configuration file is already in there and it creates a single workspace to keep your files and configurations in. You'll also want to create two files in your working directory named "ca.db.index" and "ca.db.serial". Open the "ca.db.serial" file and enter "01" on the first line. Save and close the file.

Creating a Self-signing CA

Description

A self-signing CA is a CA that you can use to sign other certificates, such as a server certificate or a personal email certificate. A self-signing CA does not require you to buy a certificate from commercial CA's such as [VeriSign](#) or [Thawte](#). The CA public key does, however, need to be installed in each browser that will use SSL communications to the server described by the certificate. Instructions for this are included in this document below.

Creating the CA

The first step is to create the CA's private key. Choose a password that would be hard to guess and keep this private key secure! Set the permissions on the file (or the directory contain the file, which is usually best since more files will go in the working directory) so that only you or an administrative group can read, write, or execute this file or files.

The following command generates an RSA private key using Triple-DES (des3) encoding using 1024 bits and placing it in a file called ca.key using a PEM format (default file format). This will prompt you for a password for the CA's private key. Remember this password and optionally keep it written down and stored in a safe place!

```
$ openssl genrsa -des3 -out ca.key 1024
```

Next, verify the contents of the file and make sure everything is correct.

```
$ openssl rsa -noout -text -in ca.key
```

The following command will create a private key without a password. (**Not recommended!**) An example reason for doing this would be for Apache web server if you enable SSL so that, during startup of the daemon, you are not prompted for a password. Use this "ca.key.unsecure" from now on instead of "ca.key". Again, keep this file secure – especially this file, since it can still be used to sign certificates (which could be used for invalid purposes) but without using a password! You will be prompted for a password after executing this command.

```
$ openssl rsa -in ca.key -out ca.key.unsecure
```

The next command will create the self-signing CA using the X.509 specification good for 365 days. You can specify a longer duration, but this allows you to create a new one after a period of time in case your old CA was cracked and used in malicious activity. Of course, you can also use a Certificate Revocation List if this happens, but that's beyond the scope of this document. The following command will prompt you for a password for the CA.

```
$ openssl req -new -x509 -days 365 -key ca.key -out ca.crt
```

Here's the questions you can expect:

```
Using configuration from /usr/local/ssl/openssl.cnf
```

OpenSSL

```
Enter PEM pass phrase:
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) [US]:
State or Province Name (full name) [Iowa]:
Locality Name (eg, city) [Ames]:
Organization Name (eg, company) [EsotericRealm]:Palisade Systems, Inc.
Organizational Unit Name (eg, section) []:IT
Common Name (eg, YOUR name) []:PalisadeSystems
Email Address []:webmaster@palisadesys.com
```

You can view the results of the last command to check for any errors and repeat the process above if necessary. You want to make sure you CA is correct because if you have signed certificate requests after this and make a change to the CA, you have to resign all your certificate requests and distribute the resulting certificates again.

```
$ openssl x509 -noout -text -in ca.crt
```

Creating a Server Certificate

Description

The server certificate is the certificate used for secure communications via SSLv3 using the HTTPS protocol, or the secure HTTP protocol. When the browser goes to a secure site, the public key is sent to the browser that is used to encrypt data that is sent to the server and decrypted by the server's private key.

Creating the Certificate Request

A certificate request is information provided by the client to give to the server to be signed. You will be prompted for a password to your private key. Again, keep this file secure!

The following command will generate your RSA private key using Triple-DES (des3) encoding with 1024 bits and place it in a file called "server.key":

```
$ openssl genrsa -des3 -out server.key 1024
```

View the output of the file and make sure everything is correct (you will be prompted for your password):

```
$ openssl rsa -noout -text -in server.key
```

If you need to create a private key without a password for use with a daemon like Apache web server, do the following command and use the key "server.key.unsecure" instead of "server.key"

OpenSSL

from now on. Again, **this is not recommend!** You will be prompted for your certificate password.

```
$ openssl rsa -in server.key -out server.key.unsecure
```

Now you create your certificate request to submit to the CA to be signed. You will be prompted for your password as well as several options. Keep your organization name the same as the CA's organization name and set the CommonName to your Fully Qualified Domain Name, or FQDN, such as "www.palisadesys.com".

```
$ openssl req -new -key server.key -out server.csr
```

Here's what you can expect to be asked:

```
Using configuration from /usr/local/ssl/openssl.cnf
Enter PEM pass phrase:
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) [US]:
State or Province Name (full name) [Iowa]:
Locality Name (eg, city) [Ames]:
Organization Name (eg, company) [EsotericRealm]:Palisade Systems, Inc.
Organizational Unit Name (eg, section) []:IT
Common Name (eg, YOUR name) []:www.palisadesys.com
Email Address []:webmaseter@palisadesys.com

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

You now have a certificate request and can move on to sign the certificate request with your CA in the next section.

Signing Your Certificate with You CA

Description

Now you have to sign your certificate using your CA. You need your CA password to do this section. You will also follow these instructions to sign a personal email certificate, so keep these instructions in mind or refer to them later when you're ready to sign a personal email certificate or code signing certificate.

Signing Your Certificate

Now you are ready to sign your certificate. The following command will sign your key using the CA password, write to the file server.crt (the certificate), and mark it valid for 365 days. Again, you can change the days value but keeping it good for 365 days lets you not have to worry about a cracked key for years on end if you don't use a Certificate Revocation List (CRL).

```
$ openssl ca -days 365 -cert ca.crt -keyfile ca.key -in server.csr -out server.crt
```

You can expect to see an output like this:

```
Using configuration from /usr/local/ssl/openssl.cnf
Loading 'screen' into random state - done
Enter PEM pass phrase:
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName           :PRINTABLE:'US'
stateOrProvinceName   :PRINTABLE:'Iowa'
localityName          :PRINTABLE:'Ames'
organizationName      :PRINTABLE:'Palisade Systems, Inc.'
organizationalUnitName:PRINTABLE:'IT'
commonName            :PRINTABLE:'www.palisadesys.com'
emailAddress          :IA5STRING:'webmaster@palisadesys.com'
Certificate is to be certified until Apr  6 18:15:38 2002 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
```

Your server certificate is now in the file server.crt. See the section on [Installing the Certificates](#) later in this document to see how to install the files in a typical environment.

Creating a Personal Email Certificate

Description

You can also create a certificate for use with encrypting, decrypting and signing email message. This uses the S/MIME standard and works very much like creating a server certificate above. The only thing different is that you use your name (ex, "Heath Stewart") for the CommonName field and your email address (ex, "hstewart@palisadesys.com") for the EmailAddress field.

Creating Your Personal Email Certificate

As described in the Description section above, the only thing different from creating a server key is the what you enter for the CommonName and EmailAddress fields. Please follow the directions above and enter your name and email address, respectively, in the fields above. Use a

filename similar to “username.mail.*” to differentiate between the certificates, keys, and requests. For example, I used “hstewart.mail.crt” for my certificate filename.

Exporting Your Personal Email Certificate

Description

Now you must export your email certificate into a format that includes both your personal and private keys. Your public key is used by others to encrypt message to you and verify your signatures. Your private key is used by you to decrypt messages send to you and sign your email. The instructions below tell you how to use OpenSSL to package these two together. See the section on [Installing the Certificates](#) later in this document.

Packaging Your Keys for Importing Later

To package your keys, you use OpenSSL that will create the keys in the right format, namely PKCS#12, which both Internet Explorer and Netscape use for personal certificates. If you would like to export just your public key in DER format so you can others the key explicitly, see the section following this one.

The following command exports a PKCS#12 format file using your personal email certificate and private key, and assigns a friendly name to it using your name. The output file is encrypted using Triple-DES (des3) encryption. You will be prompted for your private key’s password that you entered above, as well as a password for the encrypted output file. I would recommend that you use a different password for this. You will later use the second password when importing into a mail program. The second password is only for importing. Your private key still uses the first password if you are ever prompted for it in the future (depending on your mail reader configuration, which is beyond the scope of this document).

```
$ openssl pkcs12 -export -out username.mail.p12 -des3 -in username.mail.crt -inkey username.mail.key -name "Your Name"
```

To export only your public key to a common DER format (used by web browsers, email readers, and more common applications), use the following command:

```
$ openssl x509 -in username.mail.crt -outform DER -out username.mail.cer
```

You can now post this file in your personal directory on a web server or send it as an attachment. This is only your public key and does not contain your private key information that is important to keep secure. Your PKCS#12 file is the one you want to keep secure and that contains your private key. See the section [Installing the Certificates](#) later in this document to see how to install them.

Installing the Certificates

Description

To use the certificates you have created, you must install them appropriately. Server certificates are installed on the server. Since each server is different, please consult the documentation for the server on how to set it up.

- Apache..... <http://httpd.apache.org/docs/>
- IIS <http://www.microsoft.com/windows2000/library/resources/iishelp.asp>

Installing Self-Signing CA's

If you have created your own CA following the instructions above, you need to make sure that the public certificate is installed on machines that will connect via SSL, use your personal email certificates in some way, or download your signed code.

You will need to export your public key from PEM to DER format using the following command. You will be prompted for your CA's password.

```
$ openssl rsa -in ca.crt -outform DER -out ca.cer
```

Internet Explorer

To install your CA public certificate on clients with Internet Explorer, post the "ca.cer" on your web server and instruct your clients to download it by clicking the hyperlink to the "ca.cer" file and selecting **Open** when prompted. The certificate will open with a window like Figure 1:



Figure 1: Uninstalled CA Certificate Properties

Click the button labeled **Install Certificate**. You should see a wizard window like Figure 2:



Figure 2: Certificate Wizard Window

Keep clicking the **Next** button, accepting the defaults, until you get to the last page. Then, click the **Finish** button. Finally, click **Yes** when asked to add the certificate and you're finished.

Netscape

To install a self-signing CA into Netscape, go to a web server via SSL that used the CA to in signing the certificate. You will be prompted like with a window like in Figure 3:



Figure 3: Initial Netscape window for a new CA

Keep clicking the **Next** button till you see a screen like Figure 4:



Figure 4: Accepting the certificate until it expires

OpenSSL

Choose **Accept this certificate forever (until it expires)** and keep clicking the **Next** button and finally the **Finish** button. If you see a screen like Figure 5, you may have configured something wrong when creating your server certificate and you should remake it.



Figure 5: Site name does not match site on certificate

Server Certificates

Basically, you will need both your server.key and server.crt files. Apache uses each file individually in the httpd.conf file but Internet Information Server (IIS) requires them together and installed on the machine through the IIS Manager interface. You can export both your private key and your public key in the PKCS#12 format using the instructions contained within the sub-section [Packaging Your Keys for Importing Later](#) above.

Personal Email Certificates

There are many ways to do this for many different packages, but this document will cover Outlook Express and Netscape Communicator.

Outlook Express

Before you can use your certificate in Outlook Express, you must first import it into Windows. The easiest way to do this is through Internet Explorer.

OpenSSL

Open Internet Explorer and click the **Tools**→**Internet Options** menu. Click the **Content** tab, followed by the **Certificates...** button. You should see this in Figure 6 below:

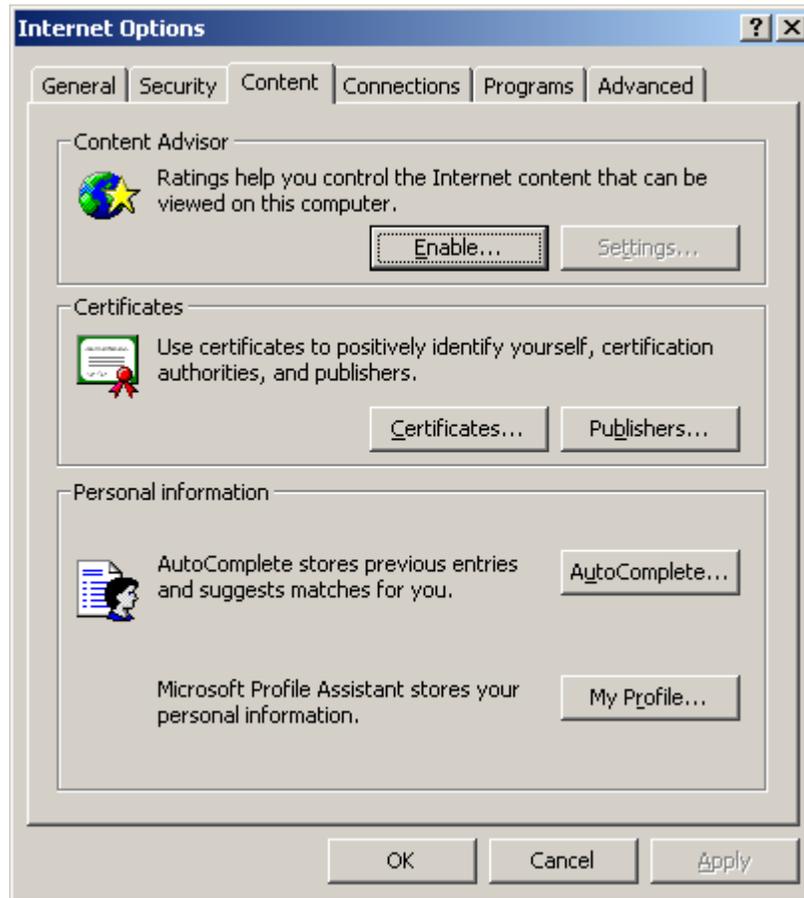


Figure 6: The Content tab in Internet Explorer

Now you should see a window like Figure 7 below:

OpenSSL

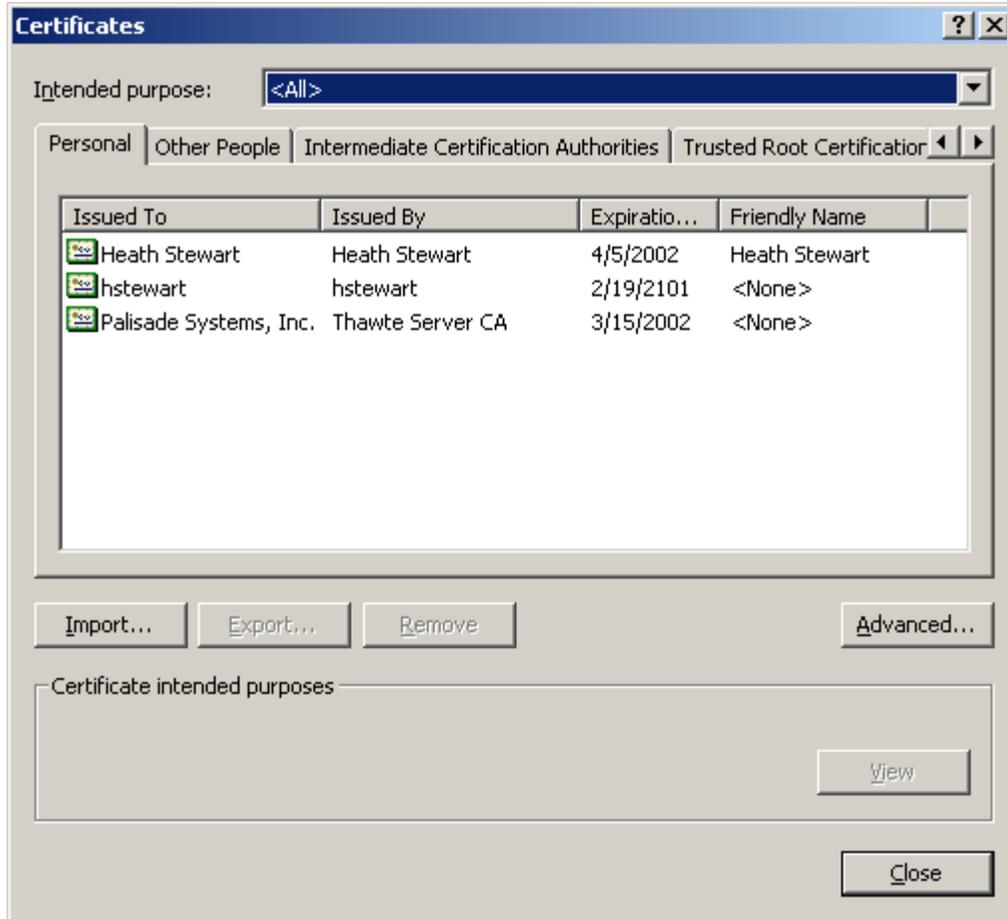


Figure 7: The Certificate Manager

Click the **Import...** button and you are presented with a wizard similar as the one earlier in this document while we were importing the CA. [See Figure 2](#). Follow the directions through the wizard till you come to the point when it asks you for a filename. Browse and find your "username.mail.p12" file and click **Next**. It will ask you for a password. Enter your second password you used when you exported your keys to a PKCS#12 format file (your "username.mail.p12" file). Also check **Mark the private key as exportable** in case you ever want to export your keys to a new machine. See Figure 8 below:

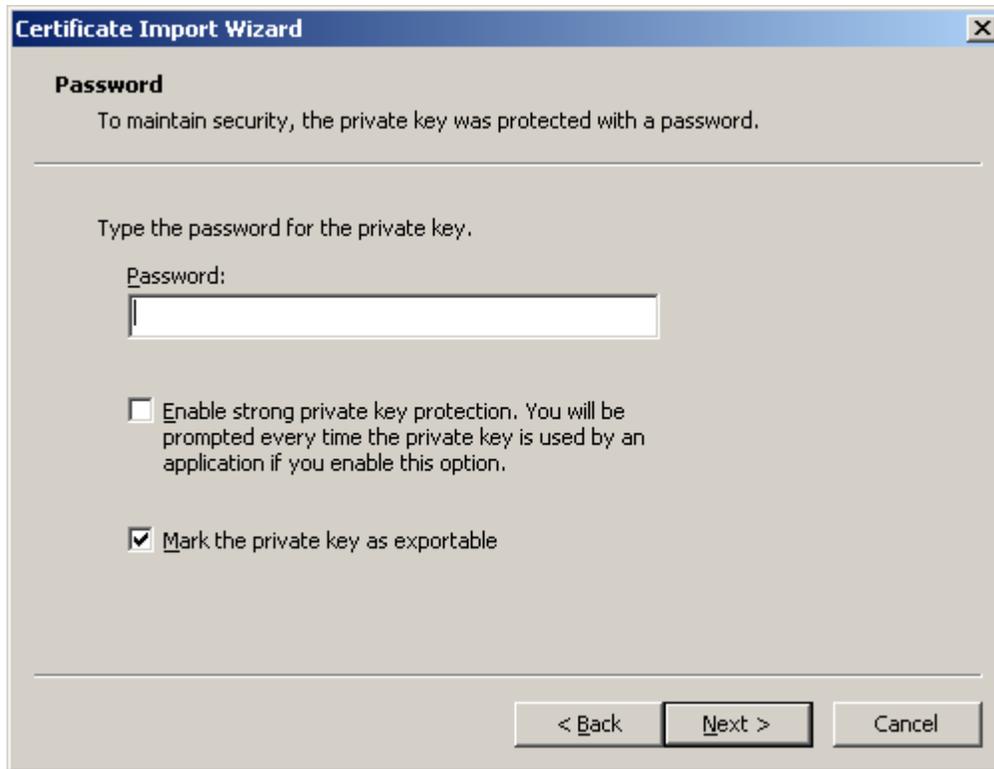


Figure 8: Entering your second password and checking options

Keep this file in a same place! Even though it is password protected with Triple-DES encryption (currently one of the best encryption algorithms), once a cracker has the file he or she can try to crack it all they want. Keep clicking the **Next** button till you see the final screen, and click the **Finish** button. Go ahead and close all the windows you just opened.

Now you're ready to configure Outlook Express with your personal email certificate. Start Outlook Express, then click the **Tools→Accounts...** menu. Double-click the account for which you want to assign the certificate. In our example using "Palisade Systems, Inc.", we will want to click the account that uses the "palisadesys.com" mail server. Next, click on the **Security** tab and you should see a property window like Figure 9 below:

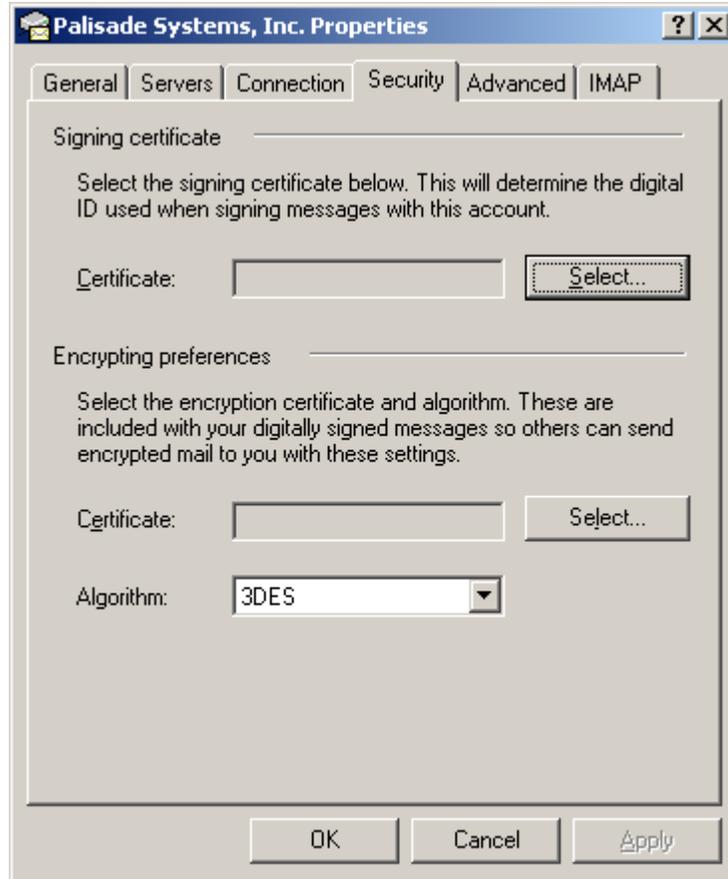


Figure 9: Outlook Express Security tab

Click both **Select** buttons in succession and select your key that you created above. The email addresses must match or signing and decrypting will be impossible. After that, you should see your friendly name that you specified above while signing your key in the **Certificate:** textboxes. Click **OK** and you are ready to use your personal email certificates.

When you create a new email, you can use the **Tools→Encrypt** and **Tools→Digitally sign** menus in the email message window. Please note, your recipient(s) must have a public key for you to encrypt a message destined for them. You must also have that key installed on your computer. You can do that following the steps above. The key will automatically be installed in the proper place. If you like, you can also set these tools as default in the options, available in your **Tools→Options...** dialog, and under your **Security** tab at the bottom of the page, like in Figure 10:

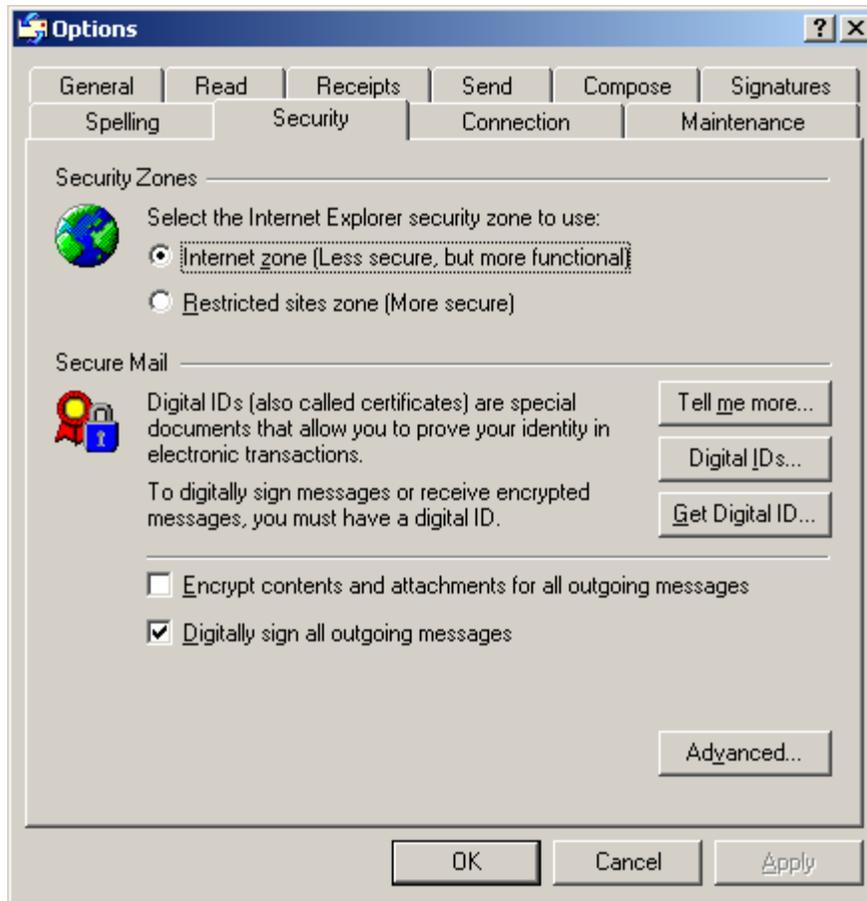


Figure 10: Setting default options for encrypting and signing

Now you are finished and can begin sending secured and valid email messages.

NOTE: The instructions for Outlook Express are very similar to Outlook and you should have no problems figuring it out now.

Netscape Communicator

Open Netscape Communicator and click the **Communicator**→**Tools**→**Security Info** menu. Under the **Certificates** section, click **Yours**. You should see a window like in Figure 11 below:

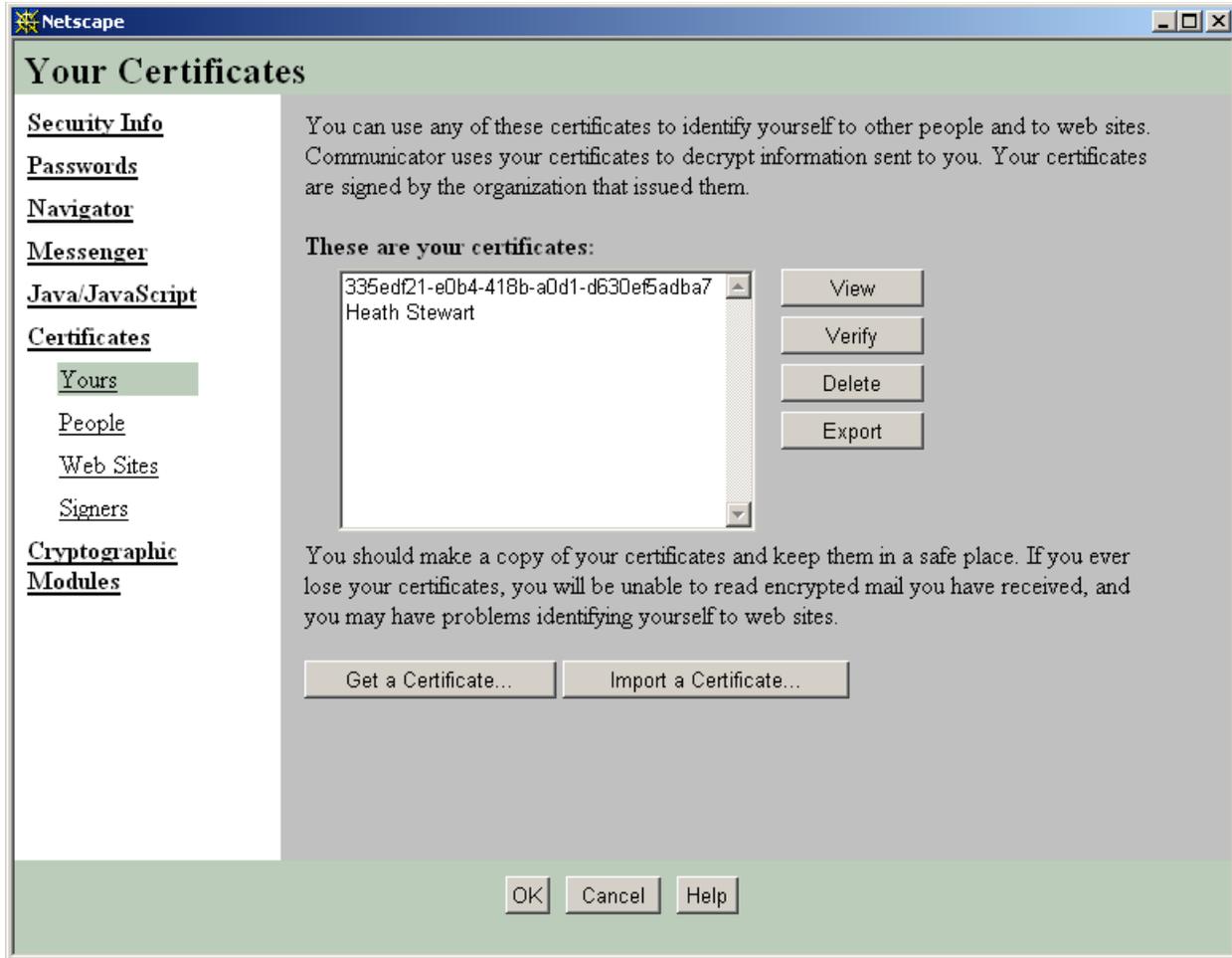


Figure 11: Netscape personal certificates

Click the **OK** button and you're ready to sign and encrypt email messages. You don't **have** to set more options (although it wouldn't hurt) because Netscape does a good job about automatically selecting the right key and creating secure communications with email servers when possible. To sign or encrypt your messages, click on the **View→Options** menu and check **Encrypt** or **Sign**, depending on what you want to do.

Summary

You can do a lot with SSL certificates. Above is just a few ways you can create them and use them. You created your own CA above but you could purchase a CA from a commercial vendor. This allows you to establish secure HTTP connections or distribute signed code without having to install your CA on all possible clients. This is just a good way to test or good to use for internal operations in a small company.

You can also install and use certificates different ways, too. The instructions above just guide you through the basic steps for Outlook Express and Netscape Communicator, but the ideas are about the same for other email readers and it shouldn't take you long to be able to send and receive encrypted and signed messages.

Appendix A: Configuration

Below is an example configuration file with defaults set. For more information on the configuration file format, see <http://www.openssl.org/docs/apps/openssl.html>.

```
#
# 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 = .                          # Where everything is kept
certs = $dir                     # Where the issued certs are kept
crl_dir = $dir                   # Where the issued crl are kept
database = $dir/ca.db.index     # database index file.
new_certs_dir = $dir            # default place for new certs.

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

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

OpenSSL

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

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 extensions 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
```

OpenSSL

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

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

stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Iowa

localityName = Locality Name (eg, city)
localityName_default = Ames

0.organizationName = Organization Name (eg, company)
0.organizationName_default = Palisade Systems, Inc.

# 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 = 40

# 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
```

OpenSSL

```
# 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=`eyed,issuer:always

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy

# 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=`eyed: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
```

OpenSSL

```
# 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=`eyed:always,issuer:always`
```

Acknowledgements

1. mod_ssl: The Apache Interface to OpenSSL. <http://www.modssl.org>.
For their great examples of some commands contained within this document and their great F.A.Q.'s that fixed some errors I was getting while signing.
2. OpenSSL Project. <http://www.openssl.org>.
For not only their great tool, openssl, but their continued support of the standards and their great online man-pages, which are a lot easier to search through than standard *nix man-pages.
3. Bridges, Stephanie. Palisade Systems, Inc.
For discussing ideas and sharing information so we could setup our servers with SSL for many different applications.