SSL Certificates HOWTO

Franck Martin

Revision History

Revision v0.1	2001−11−18	Revised by: fm

A first hand approach on how to manage a certificate authority (CA), and issue or sign certificates to be used for secure web, secure e−mail, or signing code and other usages.

Table of Contents

Chapter 1. Generalities

1.1. History

V0.1 − Franck Martin <franck@sopac.org>

Creation of the HOWTO

1.2. Introduction

Dear reader, like myself you have read intensively the man pages of the applications of the OpenSSL project, and like myself, you couldn't figure out where to start, and how to work securely with certificates. Here is the answer to most of your questions.

This HOWTO will also deal with non−linux applications, as there is no use to issue certificates if you can't use them... May be all applications won't be listed here, but please send me additional paragraphs and corrections. I can be reached at the following address:franck@sopac.org.

1.2.1. Disclaimer and Licence

1.2.2. Prior knowledge

As indicated in the introduction, this documents is an hand−on HOWTO, and it is therefore required that you consult the man pages of the OpenSSL software, as well as to read security books to learn how your security could be compromised. As certificates are meant to increase the security of your transactions, it is VERY important that you understand all the security implications of your actions and what security OpenSSL does not provide.

1.3. What is SSL and what are Certificates?

The Secure Socket Layer protocol was created by Netscape to ensure secure transactions between web servers and browsers. The protocol use a third party, a Certificate Authority (CA), to identify one end or both end of the transactions. This is in short how does it work.

1. A browser request a secure page (usually https://).
2. The web server send its public key with its certificate.
3. The browser check that the certificate was issued by a trusted party (us−ally a trusted root CA), that the certificate is still valid and that the certificate is related to the site contacted.
4. The browser then use the public key, to encrypt a random symmetric encryption key and sends it to the server with the encrypted URL required and other encrypted http data.
5. The web server decrypts the symmetric encryption key using its private key, uses the symmetric key to decrypt the URL and http data.
6. The web server sends back the requested html document and http data encrypted with the symmetric key.
7. The browser decrypt the http data and html document using the symmetric key and displays the information.

Several concepts have to be understood here.

1.3.1. Private Key/Public Key:

The encryption using a private key/public key pair ensure that the data can encrypted by one key can only be decrypted by the other key. This is sometime hard to understand, but believe me it works. The keys are similar in nature and can be used in alternatively. The key pair is based on prime numbers and their length in terms of bits ensure the difficulty of being able to decrypt the message. the trick in a key pair is to keep one key secret (the private key) and to distribute the other key (the public key) to everybody. This ensures that anybody can send you an encrypted message, that only you will be able to decrypt. You are the only one to have the other key pair, right? In the opposite , you can certify that a message is only coming from you, because you have encrypted it with you private key, and only the associated public key will decrypt it correctly. Beware, in this case the message is not secure you have just signed it. Everybody has the public key, remember!

1.3.2. The Certificate:

How do you know that you are dealing with the right person or rather the right web site. Well, someone has taken great length (if they are serious) to ensure that the web site owners are who they claim to be. This someone, you have to implicitly trust, you have his/her certificate loaded in your browser. A certificate, contains information about the owner of the certificate, like e−mail address, owner's name, certificate usage, duration of validity, and resource location or Distinguished Name (DN) which includes the Common Name (CN) (web site address or e−mail address depending of the usage) and the certificate ID of the person who certify (sign) this information. It contains also the public key and finally a hash to ensure that the certificate has not been tampered with. As you made the choice to trust the person who sign this certificate, therefore you also trust this certificate. This is what is called a certificate trust tree. Usually your browser or application has already loaded the root Certificate of well known Certification Authorities (CA) or root CA Certificate. The CA maintains a list of all signed certificates as well as a list of revoked certificates. A certificate is insecure until it is signed, as only a signed certificate cannot be modified. You can sign certificate using itself, it is called a self signed certificate. All root CA certificates are self signed.

1.3.3. The Symmetric key:

Well any Private Key/Public Key encryption algorithm is great, but it is usually not practical. It is asymmetric because you need the other key pair to decrypt. You can't use the same key to encrypt and decrypt. An algorithm using a symmetric key is much faster in doing its job than an asymmetric algorithm. But a symmetric key is potentially highly insecure. If the enemy gets hold of the key then you have no more secret information. The problem is that you must transmit the key to the other party without the enemy getting its hands on it. And as you know, nothing is secure on the Internet. The solution, is to encapsulate the symmetric key inside a message encrypted with an asymmetric algorithm. As you never transmitted your private key to anybody, then the message encrypted with the public key is secure (relatively secure, nothing is certain except death and taxes). The symmetric key is also chosen randomly, so that if the symmetric secret key is discovered then the next transaction will be totally different.

1.3.4. Encryption algorithm:

There are several encryption algorithms available, using symmetric or asymmetric methods, with keys of various lengths. Usually, algorithms cannot be patented, if Henri Poincare had patented its algorithms, then he would have been able to sue Albert Einstein... So algorithms cannot be patented except in USA. OpenSSL is developed in a country where algorithms cannot be patented and where encryption technology is not reserved to state agencies like military and secret services. During the negotiation between browser and web server, the application will indicate to each other a list of algorithm that can be understood by order of preference. The common preferred algorithm is then chosen. OpenSSL can be compiled with or without certain algorithms, so that it can be used in many countries where restrictions apply.

1.3.5. The Hash:

A hash is a number given by a hash function. This is a one way function, it means that it is impossible to get the original message knowing the hash, however the hash will drastically change even for the slightest modification in the message. It is therefore extremely difficult to modify a message without modify its hash. Hash function are used in password mechanism, in certifying applications are original (MD5 sum), and in general in ensuring that any message has not been tampered with.

1.3.6. Signing:

Signing a message, means authentifying that you have yourself assured the authenticity of the message. The message can be a text message, or someone else certificate. To sign a message, you create its hash, and then encrypt the hash with your private key, you then add the encrypted hash and your signed certificate with the message. The recipient will recreate the message hash, decrypts the encrypted hash using your well known public key stored in your signed certificate, check that both hash are equals and finally that it trusts the certificate.

1.3.7. PassPhrase:

A passprase is like a password except it is longer. In the early days passwords on Unix system were limited to 8 characters, so the term passphrase for longer passwords. Longer is the password more difficult it is to guess it. Nowadays Unix systems use MD5 hashes which have no limitation in length of the password.

1.4. What about S/Mime or other protocols?

If SSL was developed for web servers, it can be used to encrypt any protocol by encapsulating it inside SSL. This is used in IMAPS, POPS, SMTPS,... These secure protocols will used a different port than their insecure version. SSL can also be used to encrypt any transaction, there is no need to be in direct (live) contact with the recipient. S/Mime is such protocol, it encapsulate an encrypted message inside a standard e−mail. The message is encrypted using the public key of the recipient. If you are not online with the recipient then you must know its public key. Either you get it from its web site, from a repository, or you request the recipient to e−mail you its public key and certificate (to ensure you are speaking to the right recipient).

In a reverse order, the browser can send its own signed certificate to the web server, as a mean of authentication. But everybody can get the browser certificate on the CA web site. Yes, but the signed certificate has been sent encrypted with the private key, that only the public key can decrypt.

Chapter 2. Certificate Management

2.1. Installation

Nowadays, you do not have to worry too much about installing OpenSSL, as most distributions use package management applications. Refer to your distribution documentation, or read the README and INSTALL file inside the OpenSSL tarball.

I describe here some standard installation options which are necessary to know in case your installation differ.

The directory for all OpenSSL certificates is /var/ssl/. All commands and paths in this document are issued from this directory, it is not obligatory but it will help the examples.

OpenSSL by default look for a configuration file in /usr/lib/ssl/openssl.cnf so always add −config /etc/openssl.cnf to the commands openssl ca or openssl req for instance if the configuration file is located elsewhere. I use /etc/openssl.cnf so my configuration files are all in /etc.

Utilities and other libraries are located in /usr/lib/ssl.

Ensure that the utility CA.pl is in an accessible directory such as /usr/sbin. CA.pl can be found inside /usr/lib/ssl directories. CA.pl is a utility that hides the complexity of the openssl command.

/usr/sbin/CA.pl needs to be modified to include −config /etc/openssl.cnf in ca and req calls.

```
#$SSLEAY_CONFIG=$ENV{"SSLEAY_CONFIG"}
$SSLEAY_CONFIG="−config /etc/openssl.cnf";
```

/etc/openssl.cnf must be configured appropriately to minimize input entry.

```
/etc/openssl.cnf example
```

To create a certification authority use the command:

```
CA.pl −newca
```

2.2. Create a Root Certification Authority Certificate.

```
CA.pl −newcert
(openssl req −config /etc/openssl.cnf−new −x509 −keyout newreq.pem −out newreq.pem −days 365)
```

creates a self signed certificate (for Certificate Authority). The resulting file goes into newreq.pem. For the common Name (CN) use something like ACME root Certificate . This file needs to be split into 2 files cacert.pem and private/cakey.pem. The part −PRIVATE KEY− goes into private/cakey.pem while the part −CERTIFICATE− goes into cacert.pem. Delete newreq.pem when finished.

Now ensure that the file index.txt is empty and that the file serial contains 1.

You may want to increase the number of days so that your root certificate and all the certificates signed by this root does not have to be changed when the root certificate expires. I think professional companies work over 5 years for their root certificates.

```
openssl req −config /etc/openssl.cnf−new −x509 −keyout newreq.pem −out newreq.pem −days 1825
```

Now ensure that this self signed root certificate is used only to sign other certificates. The private key is highly sensible, never compromise it, by removing the passphrase that protects it.

Now you have a root Certification Authority. Other people need to trust your self−signed root CA Certificate, and therefore download it and register it on their browser.

You will have to type the passphrase each time you want to sign another certificate with it.

2.3. Create a non root Certification Authority Certificate.

FIXME because I'm not sure about the procedure.

It is possible to use any signed certificate to sign any other certificate, provide that the certificate is valid and has been issued with the signing capability. So you can create a certificate request and a private key, make the certificate been signed by a third party and install the signed certificate and private key. The part −PRIVATE KEY− goes into private/cakey.pem while the part −CERTIFICATE− goes into cacert.pem.

2.4. Install the CA root certificate as a Trusted Root Certificate

First strip the certificate from all its text to keep only the −CERTIFICATE− section

```
openssl x509 −in cacert.pem −out cacert.crt
```

Place this file on your web site as http://mysite.com/ssl/cacert.crt. Your web server has a mime entry for .crt files. Your certificate is ready to be downloaded by browser and saved on to their hard drive.

2.4.1. In Netscape

FIXME

2.4.2. In Galeon

FIXME

2.4.3. In Internet Explorer

With your browser, point to the address of the certificate and save the file on your disk. Double click on the file and the Certificate Installation wizard will start. Because the certificate is self signed, Internet explorer will automatically install it in the Trusted root Certificate Authority list. From now own Internet Explorer won't complain and any Certificate signed with this root CA Certificate will be trusted too.

2.5. Certificate management

2.5.1. Generate and Sign a certificate request

```
CA.pl −newreq
(openssl req −config /etc/openssl.cnf −new −keyout newreq.pem −out newreq.pem −days 365)
```

creates a new private key and a certificate request and place it as newreq.pem.

```
CA.pl −sign
(openssl ca −config /etc/openssl.cnf −policy policy_anything −out newcert.pem −infiles newreq.pem
```

will sign the request using the cacert.pem and commit the certificate as newcert.pem. You will need to enter the passphrase of the cacert.pem (your CA Certificate). The file newcerts/xx.pem will be created and index.txt and serial will be updated.

You private key is in newreq.pem −PRIVATE KEY− and your certificate is in newcert.pem −CERTIFICATE−

A copy of newcert.pem is placed in newcerts/ with an adequate entry in index.txt so that a client can request this information via a web server to ensure the authenticity of the certificate.

Beware of your newreq.pem file, because it contains a certificate request, but also your private key. The −PRIVATE KEY− section is not required when you sign it. So if you request someone else to sign your certificate request, ensure that you have removed the −PRIVATE KEY− section from the file. If you sign someone else certificate request, request from this person its −CERTIFICATE REQUEST− section not its private key.

2.5.2. Revoke a certificate

To revoke a certificate simply issue the command:

```
openssl −revoke newcert.pem
```

The database is updated and the certificate is marked as revoked. You need now to general the new revoked list of certificates:

```
openssl ca −gencrl −out crl/crl.pem
```

2.5.3. Renew a certificate

The user sends you its old certificate request or create a new one based on its private key.

First you have to revoke the previous certificate and sign again the certificate request.

To find the old certificate, look in the index.txt file for the Distinguished Name (DN) corresponding to the request. Get the serial Number <xx>, and use the file cert/<xx>.pem as certificate for the revocation procedure.

You may want to sign the request manually because you have to ensure that the start date and end date of validity of the new certificate are correct.

```
openssl ca −config /etc/openssl.cnf −policy policy_anything −out newcert.pem −infiles newreq.pem
```

replace [now] and [previous enddate+365days] by the correct values.

2.5.4. Build your web based Certificate Authority

There are a few requirements when you are a Certificate Authority (CA):

1. You must publish your root CA Certificate, so that it can be installed in applications.
2. You must publish the revocation list.
3. You must display a certificate detail, provided its serial number
4. You must provide a form for users to submit certificate requests.

All these requirements can be done using a web server and some scripting.

2.6. Securing Internet Protocols.

2.6.1. Use a certificate with mod_ssl in apache

First never use your self−signed root CA Certificate with any application and especially with apache as it requires you to remove the passphrase on your private key.

First generate and sign a certificate request with the Common Name (CN) as www.mysite.com.

The key needs to be made insecure, so no password is required when reading the private key. Take the newreq.pem files that contains your private key and remove the passphrase from it.

```
openssl rsa −in newreq.pem −out wwwkeyunsecure.pem
```

Because the key (PRIVATE Key) is insecure, you must know what you are doing: check file permissions, etc... If someone gets its hand on it, your site is compromised (you have been warned) Now you can use the newcert and cakeyunsecure.pem for apache.

Copy wwwkeyunsecure.pem and newcert.pem in the directory /etc/httpd/conf/ssl/ as wwwkeyunsecure.pem and wwwcert.crt respectively.

Edit /etc/httpd/conf/ssl/ssl.default−vhost.conf.

```
----
# Server Certificate:
# Point SSLCertificateFile at a PEM encoded certificate. If
# the certificate is encrypted, then you will be prompted for a
# pass phrase. Note that a kill −HUP will prompt again. A test
# certificate can be generated with `make certificate' under
# built time.
#SSLCertificateFile conf/ssl/ca.crt
SSLCertificateFile wwwcert.crt
# Server Private Key:
# If the key is not combined with the certificate, use this
# directive to point at the key file.
#SSLCertificateKeyFile conf/ssl/ca.key.unsecure SSLCertificateKeyFile wwwkeyunsecure.pem
----
```

Stop and start httpd (/etc/rc.d/init.d/httpd stop) ensure that all processes are dead (killall httpd) and start httpd (/etc/rc.d/init.d/httpd start)

2.6.2. Using a certificate with IMAPS

FIXME

2.6.3. Using a certificate with POPS

FIXME

2.6.4. Using a certificate with Postfix

FIXME

2.6.5. Generate and Sign a key with Microsoft Key Manager

In Microsoft Key Manager, Select the service you want to create a key, for instance IMAP (or WWW). Use the wizard to generate a new key. Ensure that the distinguished name won't be identical to previous generated key, for Instance for the Common Name (CN) use imap.mycompany.com. The wizard will place the request in the file C:\NewKeyRq.txt. Key Manager shows a Key with a strike to indicate the key is not signed.

Import this file in the OpenSSL /var/ssl directory rename it to newreq.pem and sign the request as usual.

```
CA.pl −sign
```

The file newcert.pem is not yet suitable for key manager as it contains some text and the −CERTIFICATE− section. We have to remove the text, the easy way is to do:

```
openssl x509 −in newcert.pem −out newcertx509.pem
```

Using a text editor is also suitable to delete everything outside the −CERTIFICATE− section.

The newcertx509.pem file contains now only the −CERTIFICATE− section.

Export the file newcertx509.pem to the Computer running key Manager and while selecting the key, right click and click on Install the Key Certificate, select this file, enter the passphrase. The key is now fully functional.

2.7. Securing E−mails.

2.7.1. Generate and use an s/mime certificate

Simply generate and sign a certificate request but with the Common Name (CN) being your e−mail address.

Now sign your message test.txt (output test.msg) using your certificate newcert.pem and your key newreq.pem:

```
openssl smime −sign −in test.tx −text −out test.msg −signer newcert.pem −inkey newreq.pem
```

2.7.2. To use this certificate with MS Outlook

You need to import it in Outlook as a pkcs12 file. To generate the pkcs12 file from your newcert.pem and newreq.pem:

```
CA.pl −pkcs12 "Franck Martin"
(openssl pkcs12 −export −in newcert.pem −inkey newreq.pem −out newcert.p12 −name "Franck Martin")
```

Beware this certificate contains your public and private key and is only secured by the passphrase. This is a file not to let into everybody's hand.

In MS Outlook go to Tools, Options and Security, Click on the import/export button select to import the newcert.p12 file, enter the export password and the Digital ID "Franck Martin" (That's my name so use your name in the above examples). And Click on Ok.

Now click on the Settings button, MS Outlook should have selected the default setting so just click on New. And finally click on Ok, except if you want to change the default settings. You are ready to send signed e−mails. When you send a signed e−mail the user at the other end will receive your public key, and will therefore be able to send you encrypted e−mails.

As you have issued this certificate from a self−signed certificate (root CA Certificate), the trust path won't be valid because the application does not know the root CA Certificate. The root CA certificate has to be downloaded and installed. Refer to the chapter "Install the CA root certificate as a Trusted Root Certificate in Internet Explorer"

2.7.3. To use this certificate with Outlook Express

FIXME

2.7.4. To use this certificate with Galeon

FIXME