SSL Certificates HOWTO

Franck Martin

Revision History

Revision v0.5	2002−10−20	Revised by: FM
Adding IPsec information from Nate Carlson, natecars@natecarlson.com / Adding IMAPS and POPS information from Bill Shirley, webnut@telocity.com / Adding WinCrypt information from Colin McKinnon, colin@wew.co.uk		
Revision v0.4	2002−06−22	Revised by: FM
Various corrections − adding ASCII Art		
Revision v0.3	2002−05−09	Revised by: FM
Adding x509v3 extension information − Correcting spelling		
Revision v0.2	2001−12−06	Revised by: FM
Adding openssl.cnf file / Adding CRL info from Averroes, a.averroes@libertysurf.fr / Correcting spelling		
Revision v0.1	2001−11−18	Revised by: FM
Creation of the HOWTO		

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

Table of Contents

Chapter 1. Generalities

1.1. Introduction

Dear reader, like myself, you have intensively read 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: there is no use to issue certificates if you can't use them... 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.

This HOWTO is published on The Linux Documentation Project this is where you will find the lastest version of this document.

1.1.1. Disclaimer and Licence

This document is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

In short, if the advises given here break the security of your e−commerce application, then tough luck− it's never our fault. Sorry.

Please freely copy and distribute (sell or give away) this document in any format. It's requested that corrections and/or comments be forwarded to the document maintainer. You may create a derivative work and distribute it provided that you:

1. Send your derivative work (in the most suitable format such as sgml) to the LDP (Linux Documentation Project) or the like for posting on the Internet. If not the LDP, then let the LDP know where it is available.
2. License the derivative work with this same license or use GPL. Include a copyright notice and at least a pointer to the license used.
3. Give due credit to previous authors and major contributors. If you're considering making a derived work other than a translation, it's requested that you discuss your plans with the current maintainer.

It is also requested that if you publish this HOWTO in hardcopy that you send the authors some samples for 'review purposes' :−). You may also want to send something to cook my noodles ;−)

1.1.2. 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. You should as well read security books to learn how your security could be compromised. 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.

1.2. 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 uses a third party, a Certificate Authority (CA), to identify one end or both end of the transactions. This is in short how it works.

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

Several concepts have to be understood here.

1.2.1. Private Key/Public Key:

The encryption using a private key/public key pair ensures that the data can be encrypted by one key but can only be decrypted by the other key pair. This is sometime hard to understand, but believe me it works. The keys are similar in nature and can be used alternatively: what one key emcrypts, the other key pair can decrypt. The key pair is based on prime numbers and their length in terms of bits ensures the difficulty of being able to decrypt the message without the key pairs. 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. 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 secured you have only signed it. Everybody has the public key, remember!

One of the problem left is to know the public key of your correspondent. Usually you will ask him to send you a non confidential signed message that will contains his publick key as well as a certificate.

```
Message-->[Public Key]-->Encrypted Message-->[Private Key]-->Message
```

1.2.2. 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 root Certificate). A certificate, contains information about the owner of the certificate, like e−mail address, owner's name, certificate usage, duration of validity, 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 certifies (signs) 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 signs this certificate, therefore you also trust this certificate. This is a certificate trust tree or certificate path. Usually your browser or application has already loaded the root certificate of well known Certification Authorities (CA) or root CA Certificates. 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 a certificate using itself, it is called a self signed certificate. All root CA certificates are self signed.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=FJ, ST=Fiji, L=Suva, O=SOPAC, OU=ICT, CN=SOPAC Root CA/Email=administrator@sopac.org
        Validity
            Not Before: Nov 20 05:47:44 2001 GMT
            Not After : Nov 20 05:47:44 2002 GMT
        Subject: C=FJ, ST=Fiji, L=Suva, O=SOPAC, OU=ICT, CN=www.sopac.org/Email=administrator@sopac.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:ba:54:2c:ab:88:74:aa:6b:35:a5:a9:c1:d0:5a:
                    9b:fb:6b:b5:71:bc:ef:d3:ab:15:cc:5b:75:73:36:
                    b8:01:d1:59:3f:c1:88:c0:33:91:04:f1:bf:1a:b4:
                    7a:c8:39:c2:89:1f:87:0f:91:19:81:09:46:0c:86:
                    08:d8:75:c4:6f:5a:98:4a:f9:f8:f7:38:24:fc:bd:
                    94:24:37:ab:f1:1c:d8:91:ee:fb:1b:9f:88:ba:25:
                    da:f6:21:7f:04:32:35:17:3d:36:1c:fb:b7:32:9e:
                    42:af:77:b6:25:1c:59:69:af:be:00:a1:f8:b0:1a:
                    6c:14:e2:ae:62:e7:6b:30:e9
                Exponent: 65537 (0x10001)
         X509v3 extensions:
             X509v3 Basic Constraints:
                 CA:FALSE
             Netscape Comment:
                 OpenSSL Generated Certificate
             X509v3 Subject Key Identifier:
                 FE:04:46:ED:A0:15:BE:C1:4B:59:03:F8:2D:0D:ED:2A:E0:ED:F9:2F
             X509v3 Authority Key Identifier:
                 keyid:E6:12:7C:3D:A1:02:E5:BA:1F:DA:9E:37:BE:E3:45:3E:9B:AE:E5:A6
                 DirName:/C=FJ/ST=Fiji/L=Suva/O=SOPAC/OU=ICT/CN=SOPAC Root CA/Email=administrator@sopac.org
                 serial:00
    Signature Algorithm: md5WithRSAEncryption
        34:8d:fb:65:0b:85:5b:e2:44:09:f0:55:31:3b:29:2b:f4:fd:
        aa:5f:db:b8:11:1a:c6:ab:33:67:59:c1:04:de:34:df:08:57:
        2e:c6:60:dc:f7:d4:e2:f1:73:97:57:23:50:02:63:fc:78:96:
        34:b3:ca:c4:1b:c5:4c:c8:16:69:bb:9c:4a:7e:00:19:48:62:
        e2:51:ab:3a:fa:fd:88:cd:e0:9d:ef:67:50:da:fe:4b:13:c5:
        0c:8c:fc:ad:6e:b5:ee:40:e3:fd:34:10:9f:ad:34:bd:db:06:
        ed:09:3d:f2:a6:81:22:63:16:dc:ae:33:0c:70:fd:0a:6c:af:
        bc:5a
-----BEGIN CERTIFICATE-----
MIIDoTCCAwqgAwIBAgIBATANBgkqhkiG9w0BAQQFADCBiTELMAkGA1UEBhMCRkox
DTALBgNVBAgTBEZpamkxDTALBgNVBAcTBFN1dmExDjAMBgNVBAoTBVNPUEFDMQww
CgYDVQQLEwNJQ1QxFjAUBgNVBAMTDVNPUEFDIFJvb3QgQ0ExJjAkBgkqhkiG9w0B
CQEWF2FkbWluaXN0cmF0b3JAc29wYWMub3JnMB4XDTAxMTEyMDA1NDc0NFoXDTAy
MTEyMDA1NDc0NFowgYkxCzAJBgNVBAYTAkZKMQ0wCwYDVQQIEwRGaWppMQ0wCwYD
VQQHEwRTdXZhMQ4wDAYDVQQKEwVTT1BBQzEMMAoGA1UECxMDSUNUMRYwFAYDVQQD
Ew13d3cuc29wYWMub3JnMSYwJAYJKoZIhvcNAQkBFhdhZG1pbmlzdHJhdG9yQHNv
cGFjLm9yZzCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAulQsq4h0qms1panB
```

```
0Fqb+2u1cbzv06sVzFt1cza4AdFZP8GIwDORBPG/GrR6yDnCiR+HD5EZgQlGDIYI
2HXEb1qYSvn49zgk/L2UJDer8RzYke77G5+IuiXa9iF/BDI1Fz02HPu3Mp5Cr3e2
JRxZaa++AKH4sBpsFOKuYudrMOkCAwEAAaOCARUwggERMAkGA1UdEwQCMAAwLAYJ
YIZIAYb4QgENBB8WHU9wZW5TU0wgR2VuZXJhdGVkIENlcnRpZmljYXRlMB0GA1Ud
DgQWBBT+BEbtoBW+wUtZA/gtDe0q4O35LzCBtgYDVR0jBIGuMIGrgBTmEnw9oQLl
uh/anje+40U+m67lpqGBj6SBjDCBiTELMAkGA1UEBhMCRkoxDTALBgNVBAgTBEZp
amkxDTALBgNVBAcTBFN1dmExDjAMBgNVBAoTBVNPUEFDMQwwCgYDVQQLEwNJQ1Qx
FjAUBgNVBAMTDVNPUEFDIFJvb3QgQ0ExJjAkBgkqhkiG9w0BCQEWF2FkbWluaXN0
cmF0b3JAc29wYWMub3JnggEAMA0GCSqGSIb3DQEBBAUAA4GBADSN+2ULhVviRAnw
VTE7KSv0/apf27gRGsarM2dZwQTeNN8IVy7GYNz31OLxc5dXI1ACY/x4ljSzysQb
xUzIFmm7nEp+ABlIYuJRqzr6/YjN4J3vZ1Da/ksTxQyM/K1ute5A4/00EJ+tNL3b
Bu0JPfKmgSJjFtyuMwxw/Qpsr7xa
-----END CERTIFICATE-----
```


As You may have noticed, the certificate contains the reference to the issuer, the public key of the owner of this certificate, the dates of validity of this certificate and the signature of the certificate to ensure this certificate hasen't been tampered with. The certificate does not contain the private key as it should never be transmitted in any form whatsoever. This certificate has all the elements to send an encrypted message to the owner (using the public key) or to verify a message signed by the author of this certificate.

1.2.3. The Symmetric key:

Well, Private Key/Public Key encryption algorithms are great, but they are not usually 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 the same key to decrypt and encrypt is deemed to have a symmetric key. A symmetric algorithm 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. You must therefore transmit the key to the other party without the enemy getting its hands on it. 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. You have 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.

```
Symetric Key-->[Public Key]-->Encrypted Symetric Key-->[Private Key]-->Symetric Key
```

1.2.4. 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 his algorithms, then he would have been able to sue Albert Einstein... So algorithms cannot be patented except mainly 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 applications will indicate to each other a list of algorithms that can be understood ranked 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.

1.2.5. The Hash:

A hash is a number given by a hash function from a message. 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 while keeping its original hash. It is also called a message digest. Hash functions are used in password mechanisms, in certifying that applications are original (MD5 sum), and in general in ensuring that any message has not been tampered with. It seems that the Internet Enginering Task Force (IETF) prefers SHA1 over MD5 for a number of technical reasons (Cf RFC2459 7.1.2 and 7.1.3).

1.2.6. Signing:

Signing a message, means authentifying that you have yourself assured the authenticity of the message (most of the time it means you are the author, but not neccesarily). The message can be a text message, or someone else's 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 check the certificate.

The other advantage of signing your messages is that you transmit your public key and certificate automatically to all your recipients.

There are usually 2 ways to sign, encapsulating the text message inside the signature (with delimiters), or encoding the message altogether with the signature. This later form is a very simple encryption form as any software can decrypt it if it can read the embedded public key. The advantage of the first form is that the message is human readable allowing any non complaint client to pass the message as is for the user to read, while the second form does not even allow to read part of the message if it has been tampered with.

1.2.7. 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 harder it is to guess. Nowadays Unix systems use MD5 hashes which have no limitation in length of the password.

1.2.8. Public Key Infrastructure

The Public Key Infrastructure (PKI) is the software management system and database system that allows to sign certifcate, keep a list of revoked certificates, distribute public key,... You can usually access it via a website and/or ldap server. There will be also some people checking that you are who you are... For securing individual applications, you can use any well known commercial PKI as their root CA certificate is most likely to be inside your browser/application. The problem is for securing e−mail, either you get a generic type certificate for your e−mail or you must pay about USD100 a year per certificate/e−mail address. There is also no way to find someone's public key if you have never received a prior e−mail with his certificate (including his public key).

1.3. What about S/Mime or other protocols?

If SSL was developed for web servers, it can be used to encrypt any protocol. Any protocol can be encapsulated inside SSL. This is used in IMAPS, POPS, SMTPS,... These secure protocols will use 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 encapsulates 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: most distributions use package management applications. Refer to your distribution documentation, or read the README and INSTALL file inside the OpenSSL tarball. I want also to avoid to make this HOWTO, an installation HOWTO rather than an HOWTO use certificates.

I describe here some standard installation options which are necessary to know for the samples following. Your installation may 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 mandatory but it will help the examples.

OpenSSL by default looks 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. I use /etc/openssl.cnf so all my configuration files are all in /etc.

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

2.1.1. The CA.pl utility

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. In all the examples, when I use CA.pl, I will also put the openssl equivalent in brakets.

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

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

2.1.2. The openssl.cnf file

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

```
#−−−Begin−−−
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#
RANDFILE  = $ENV::HOME/.rnd
oid_file  = $ENV::HOME/.oid
oid_section  = new_oids
# 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             = /var/ssl                # Where everything is kept
certs           = $dir/certs              # Where the issued certs are kept
crl_dir         = $dir/crl                # Where the issued crl are kept
database        = $dir/index.txt          # database index file.
new_certs_dir   = $dir/newcerts           # default place for new certs.
certificate     = $dir/cacert.pem         # The CA certificate
serial          = $dir/serial             # The current serial number
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
# 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= 7                       # how long before next CRL
default_md      = sha1                    # 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    = optional
localityName           = 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
default_md         = sha1
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 = FJ
countryName_min     = 2
countryName_max     = 2

stateOrProvinceName         = State or Province Name (full name)
stateOrProvinceName_default = Fiji
localityName          = Locality Name (eg, city)
localityName_default  = Suva
0.organizationName         = Organization Name (eg, company)
0.organizationName_default = SOPAC
# 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 = ITU
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
# 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  = "Certificate issued by https://www.sopac.org/ssl/"
# 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
# Copy subject details
# issuerAltName=issuer:copy
# This is the base URL for all others URL addresses
# if not supplied
nsBaseUrl  = https://www.sopac.org/ssl/
# This is the link where to download the latest Certificate
# Revocation List (CRL)
nsCaRevocationUrl = https://www.sopac.org/ssl/sopac-ca.crl
# This is the link where to revoke the certificate
nsRevocationUrl  = https://www.sopac.org/ssl/revocation.html?
# This is the location where the certificate can be renewed
nsRenewalUrl  = https://www.sopac.org/ssl/renewal.html?
# This is the link where the CA policy can be found
nsCaPolicyUrl  = https://www.sopac.org/ssl/policy.html
# This is the link where we can get the issuer certificate
issuerAltName = URI:https://www.sopac.org/ssl/sopac.crt
# This is the link where to get the latest CRL
crlDistributionPoints = URI:https://www.sopac.org/ssl/sopac-ca.crl
[ 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
# RAW DER hex encoding of an extension: beware experts only!
# 1.2.3.5=RAW:02:03
# You can even override a supported extension:
# basicConstraints= critical, RAW:30:03:01:01:FF
# This will be displayed in Netscape's comment listbox.
nsComment  = "Certificate issued by https://www.sopac.org/ssl/"
# This is the base URL for all others URL addresses
# if not supplied
nsBaseUrl  = https://www.sopac.org/ssl/
# This is the link where to download the latest Certificate
# Revocation List (CRL)
nsCaRevocationUrl = https://www.sopac.org/ssl/sopac-ca.crl
# This is the link where to revoke the certificate
nsRevocationUrl  = https://www.sopac.org/ssl/revocation.html?
# This is the location where the certificate can be renewed
nsRenewalUrl  = https://www.sopac.org/ssl/renewal.html?
# This is the link where the CA policy can be found
nsCaPolicyUrl  = https://www.sopac.org/ssl/policy.html
# This is the link where we can get the issuer certificate
issuerAltName = URI:https://www.sopac.org/ssl/sopac.crt
```

```
# This is the link where to get the latest CRL
crlDistributionPoints = URI:https://www.sopac.org/ssl/sopac-ca.crl
[ crl_ext ]
# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.
# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always,issuer:always
#----End----
```

A few comments on openssl.cnf.

- Variable names can use the suffixes _default for default value, _min for the minimum number of characters required and _max for the maximum number of characters required.
- The file is composed of [Sections] of variables.

dir:
: Specifies the base directory.

default_ca:
: Specifies which section contains the variables for a default certificate.

basicConstraints:
: Defines the usage of the certificate, for instance with CA:TRUE, the certificate is a root CA Certificate.

2.1.3. Create the Certification Authority

To create a certification authority, use the command after correctly editing openssl.cnf:

```
CA.pl -newca
```

The utility will ask you to select a certificate file to act as you CA certificate or you are prompted to create one. Follow the steps to create one as exrecise. In the next chapter we will overwrite this default created CA to create a new one with a longer life span. CA.pl creates only 365 days certificates.

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 –RSA 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 01.

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 to 10 years for their root certificates.

```
openssl req -config /etc/openssl.cnf -new -x509 -keyout private/cakey.pem \
-out cacert.pem -days 3650
```

This last command is better than "CA.pl –newcert" as it will place the files in the required locations and create a root CA valid for 10 years.

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. Some people will place the private key on a floppy and will load it only when signing other certificates. If you computer gets hacked they can't physically get hold of the private key, if it is on a floppy.

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, provided 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 should have a mime entry for .crt files. Your certificate is ready to be downloaded by any browser and saved.

It is important to publish the root CA Certificate on a web site as it is unlikely that people will have it already loaded on their browser. Beware, somebody could fake your web site and fake your root CA Certificate. If you can have more than one way for users to get your certificate, it is unlikely that a hacker will be able to corrupt everything.

Microsoft proposes a windows update feature that will push approved root certificate to internet explorers out there. You can contact Microsoft to have your root certificate added in their database and maybe in their future releases.

2.4.1. In Netscape/Mozilla

Download the certificate from the web server or from the file system using Netscape. Netscape automatically recognises that it is a root certificate and will propose you to add it in its store. Follow the wizard to install the certifcate. At the end of the wizard you have to specify for which type of application you trust this certifcate: web site security, e–mail signing, or code signing.

2.4.2. In Galeon

Galeon works like Mozilla as it uses the same HTML rendering engine. However there is no Certificate management tool included in Galeon.

2.4.3. In Opera

FIXME

2.4.4. 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 on, Internet Explorer won't complain and any Certificate signed with this root CA Certificate will be trusted too.

You can also open it from Internet explorer which will display the certificate. Click on the button Install Certificate to launch the Certificate Installation wizard.

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. Enter a Common Name (CN) the main usage of the certificate for instance www.sopac.org if you want to secure the website www.sopac.org, or enter franck@sopac.org if you want to use to secure the e−mails of franck@sopac.org.

```
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 now need to generate the new revoked list of certificates:

```
openssl ca -gencrl -config /etc/openssl.cnf -out crl/sopac-ca.crl
```

This Certificate Revokation List (CRL) file should be made available on your web site.

You may want to add the parameters crldays or crlhours and crlexts when you revoke a certificate. The first two parameters indicate when the next CRL will be updated and the last one will use the crl_exts section in openssl.cnf to produce a CRL v2 instead of a CRL v1.

```
openssl ca -gencrl -config /etc/openssl.cnf -crldays 7 -crlexts crl_ext \
-out crl/sopac-ca.crl
```

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 -startdate [now] -enddate [previous enddate+365days]
```

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

2.5.4. Display a certificate

You may have a certificate in its coded form, to read the details of the certificate just issue the following command:

```
openssl x509 -in newcert.pem -noout -text
```

2.5.5. The index.txt file

In the index.txt file you can find the various certificate managed by OpenSSL. The entries are maked with R for Revoked, V for Valid and E for expired.

2.5.6. 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 widely 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.

FIXME: some code here for the web interface...

Chapter 3. Using Certificates in Applications

3.1. Securing Internet Protocols.

3.1.1. Using 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. Remove any extra information to keep only the −−−CERTIFCATE −−− part.

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)

3.1.2. Using a certificate with IMAPS

Read the paragraph on "Using a certificate with POPS", for more information.

3.1.3. Using a certificate with POPS

A pem file for ipop3sd can be created by generating a certificate, unsecuring the private key and combining the two into /etc/ssl/imap/ipop3sd.pem. This is the location where the imap rpm on Mandrake 9.0 expects to find the file. A similar procedure can be used for imap and putting the file in /etc/ssl/imap/imapsd.pem.

The CN should be the name that the mail client connects to (e.g mail.xyz.org). In MS−Outlook, on the server tab, enter for the incoming mail server mail.xyz.org and on the Advanced tab check the "This server requires a secure connection (SSL)", this will change the connection port to 995 (imaps). The trusted root CA must be installed in MS Internet Explorer to validate the certificate from mail.xyz.org.

3.1.4. Using a certificate with Postfix

FIXME

3.1.5. Using a certificate with Stunnel

FIXME

3.1.6. Generate and Sign a key with Microsoft Key Manager

In Microsoft Key Manager, select the service you want to create a key for, 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 keys, 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 now contains only the −CERTIFICATE− section.

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

3.2. Securing E−mails.

3.2.1. 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.txt −text −out test.msg −signer newcert.pem −inkey newreq.pem
```

You can now transmit test.msg to anybody, you can use this procedure to make signed advisories, or other signed documents to be published digitally.

3.2.2. 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")
```

or use this command to bundle the signing certificate with your pkcs12 file

```
openssl pkcs12 −export −in newcert.pem −inkey newreq.pem −certfile cacert.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".

You can send your message as encrypted signed messages or clear text message. The encryption is not really an encryption as the message contains everything needed to decrypt the message, but it ensures that the recipient won't read the message if he does not have an s/mime compliant reader.

Note that early version of MS−Outlook XP will search the Internet to verify the validy of the certificate. It can take several seconds before the e−mail is displayed and several minutes for MS−Outlook XP to timeout when you don't have a full time or on−demand Internet connection. The bug is that this process is exclusive, the whole machine freezes till MS−Outlook XP has finished somehow.

3.2.3. To use this certificate with MS Outlook Express

FIXME

3.2.4. To use this certificate with Netscape Messenger

FIXME

3.2.5. To use this certificate with Evolution

Evolution 1.0 does not work with S/MIME, but only with PGP. It is planned that Evolution will handle S/MIME in a future release (from the evolution bug database). However in some instances Evolution recognises that the document is clear text signed and displays it correctly, even though it can't check the signature (early versions of Evolution does not understand one of the 3 MIME signature types, unfortunately the one MS−Outlook uses quite often).

3.2.6. To use this certificate with Balsa

FIXME

3.2.7. To use this certifcate with KMail

FIXME

3.3. Securing Files

3.3.1. WinCrypt

WinCrypt uses the Microsoft crypto API to encrypt and /or sign files. It will optionnaly create a zip archive of the selected files/folders before signing. It provides a front end to the certificate store, allowing the user to browse the installed certificate store, install and delete certificates and choose the certificate to use for WinCrypt signing.

The procedure for creating a certificate is the same as for Microsoft Outlook. Indeed it uses the same certificate store, you can point WinCrypt to a certificate previously installed for Outlook and vice−versa.

It is possible to verify a WinCrypt signed file filename.sgn using:

```
openssl smime -verify -inform der -in filename.sgn -CAfile cacert.crt
```

To sign a file with openSSL in a compatible format use:

```
openssl smime -sign -outform der -nodetach -out filename.sgn \
-signer certificate.pem -in filename.txt
```

To view the structure of a signed file:

```
openssl asn1parse -inform der -in filename.sgn
```

3.4. Securing Code

3.4.1. Micosoft Code

You can sign your programs and applet to certify that you are the author of such code. It is important for your customes to trust that nobody has tried to insert a virus or a backdoor inside your code. To authenticate your code you need Microsoft Authenticode SDK. You can get it from the Microsoft web site in the MSDN section.

Gernerate a certificate as usual but with a Common Name (CN) like "ACME Software Cert". Have the certificate signed by the CA and convert it to a pkcs12 format.

```
CA.pl -newreq
CA.pl -sign
CA.pl -pkcs12 "ACME Software Cert"
```

You get a file called newcert.p12 that you import in the Certificate store by clicking on the file when in Windows.

You can now use this certificate for signing your code

```
signcode -cn "ACME Software cert" -tr 5 -tw 2 -n "My Application" \
-i http://www.acme.com/myapp/ \
-t http://timestamp.verisign.com/scripts/timstamp.dll myapp.exe
```

When you try to install and run your application a dialog will appears with the title "My Application" and with a link pointed by the –i argument.

3.5. IPSec

IPSec is a new protocol that sits on top of IP that provides ad−hoc encrypted links between 2 hosts on the Internet. The IPSec implementation is mandatory for IPv6 and can be added to IPv4. If IPSec is part of IPv6, it does not mean that it is deployed by network managers. IPSec is not simple to implement due to the difficulty of having mechanisms to exchange keys automatically between machines. DNS can help, but it is not mainstream, and well known Certificate Authorities do not yet deliver adequate certification facilities for a wide deployement in the enterprise.

3.5.1. FreeS/WAN

FreeS/WAN is a popular implementation of IPSec for GNU/Linux. At its current version (1.9.7) it needs to be patched to incorporate X.509 capability. You can find a patched version on this site. Some GNU/Linux distrubutions have applied the patch for you so check your package. The advantage of this version is that you can use openssl to create certificates to use with FreeS/WAN and DNS CERT records, but more specifically you can interact with the Microsoft Implementation of IPSec. For more information check Nate's page.

3.5.1.1. FreeS/WAN gateway machine

Generate a certificate with the CN beeing the fully qualified domain name of your IPSec gateway: host.example.com. Do not forget to sign the certificate. You have two files newcert.pem and newreq.pem. The file newreq.pem contains the private key and some extra information therefore needs to be edited to contain only the private key. Delete everything outside the −−BEGIN RSA PRIVATE KEY−− and −−END RSA PRIVATE KEY−−. Move the files to the appropriate locations on your gateway machine. Make sure that you do that securely. On my distribution all configuration files for FreeS/WAN are located in /etc/freeswan, it could be different in yours.

```
mv newreq.pem /etc/freeswan/ipsec.d/private/host.example.com.key
mv newcert.pem /etc/freeswan/ipsec.d/host.example.com.pem
```

Copy also your root certificate to the FreeS/WAN configuration directory. Copy only the certificate, not the key.

```
mv cacert.pem /etc/freeswan/ipsec.d/cacerts
```

Generate a certificate revocation list or copy yours to the right location.

```
openssl ca −genrcl −out /etc/freeswan/ipsec.d/crls/crl.pem
```

Still on the gateway machine, configure the ipsec.secrets file by including the line:

```
: RSA host.example.com.key "password"
```

The password being the one used to generate the key pair. Configure ipsec.conf as following:

```
config setup
interfaces=%defaultroute
klipsdebug=none
plutodebug=none
plutoload=%search
plutostart=%search
uniqueids=yes
conn %default
keyingtries=1
compress=yes
disablearrivalcheck=no
authby=rsasig
leftrsasigkey=%cert
rightrsasigkey=%cert
conn roadwarrior−net
leftsubnet=<your_subnet>/<your_netmask>
also=roadwarrior
conn roadwarrior
right=%any
left%defaultroute
leftcert=host.example.com.pem
auto=add
pfs=yes
```

As you can see there are 2 connections beeing established, one to the gateway machine and one to the network behind the gateway machine. This is particulary useful if you are operating some kind of firewall/NAT on your gateway machine. The configuration is such that anybody with a valid certificate will be able to connect to the gateway machine.

3.5.1.2. FreeS/WAN client machine

The procedure is similar, you need to generate a certificate for the client machine with the CN being the fully qualified domain name of the client machine, for instance clienthost.example.com. This certificate must be signed by the same signing authorithy as the gateway certificate. This is how the link will be authorised.

As with the gateway copy the following files securely to the configuration directories:

```
mv newreq.pem /etc/freeswan/ipsec.d/private/clienthost.example.com.key
mv newcert.pem /etc/freeswan/ipsec.d/clienthost.example.com.pem
```

Copy also your root certificate to the FreeS/WAN configuration directory. Copy only the certificate, not the key.

```
mv cacert.pem /etc/freeswan/ipsec.d/cacerts
```

Generate a certificate revocation list or copy yours to the right location.

```
openssl ca -genrcl -out /etc/freeswan/ipsec.d/crls/crl.pem
```

Finally you need to copy also the certificate (not the private key) of your gateway machine

```
mv host.example.com.pem /etc/fresswan/ipsec.d/host.example.com.pem
```

Similarly edit your ipsec.secrets file to load the client private key

```
: RSA clienthost.example.com.key "password"
```

and edit the ipsec.conf as follows to enable the connection:

```
config setup
interfaces=%defaultroute
klipsdebug=none
plutodebug=none
plutoload=%search
plutostart=%search
uniqueids=yes
conn %default
keyingtries=0
compress=yes
disablearrivalcheck=no
authby=rsasig
leftrsasigkey=%cert
rightrsasigkey=%cert
conn roadwarrior-net
left=(ip of host)
leftsubnet=(gateway_host_subnet)/(gateway_host_netmask)
also=roadwarrior
conn roadwarrior
left=(ip of host)
leftcert=host.example.com.pem
right=%defaultroute
rightcert=clienthost.example.com.pem
```

```
auto=add
pfs=yes
```

Now you can start the VPN link

```
ipsec auto --up roadwarrior
ipsec auto --up roadwarrior-net
```

To start the link automatically, replace in the configuration file 'auto=add' by 'auto=start'

3.5.1.3. MS Windows 2000/XP client machine

The procedure is the same as for the FreeS/WAN client. Generate a certificate with a CN of winhost.example.com, but you will have to convert this certificate into a .p12 file. Follow the procedure in the chapter "Using this certificate with MS−Outlook" but ensure that the .p12 file is bundled with the root CA certificate: winhost.example.com.p12

Additionally note the output of:

```
openssl x509 -in cacert.pem -noout -subject
```

Copy this file securely to the MS−Windows machine.

You know need to install Marcus Muller's ipsec.exe utility in for instance c:\ipsec directory.

Open Microsoft Management Console (MMC), in 'Add/Remove Snap−in' click on 'Add' then click on 'Certificates', then 'Add' Select 'Computer Account', and 'Next'. Select 'Local computer', and 'Finish'. Click on 'IP Security Policy Management', and 'Add'. Select 'Local Computer', and 'Finish' click 'Close' then 'OK'

Now you can add the .p12 certificate

Click the plus arrow by 'Certificates (Local Computer)' then right−click 'Personal', and click 'All Tasks' then 'Import' click 'Next'. Type the path to the .p12 file (or browse and select the file), and click 'Next'. Type the export password, and click 'Next'. Select 'Automatically select the certificate store based on the type of certificate', and click 'Next'. Click 'Finish', and say yes to any prompts that pop up. Exit the MMC, and save it as a file so you don't have to re−add the Snap In each time.

Install ipsecpol.exe (Windows 2000) or ipseccmd.exe (Windows XP) as described in the documentation for the ipsec utility. Edit your ipsec.conf (on the windows machine), replacing the "RightCA" with the output of the 'openssl x509 −in cacert.pem −noout −subject'; reformatted as below (you need to change the /'s to commas, and change the name of some of the fields −− just follow the example below):

```
conn roadwarrior
left=%any
right=(ip_of_remote_system)
rightca="C=FJ, ST=Fiji, L=Suva, O=SOPAC, OU=ICT, CN=SOPAC Root"
network=auto
auto=start
pfs=yes
conn roadwarrior-net
left=%any
right=(ip_of_remote_system)
rightsubnet=(your_subnet)/(your_netmask)
```

```
rightca="C=FJ, ST=Fiji, L=Suva, O=SOPAC, OU=ICT, CN=SOPAC Root"
network=auto
auto=start
pfs=yes
```

Start the link

Run the command 'ipsec.exe'. Here's example output:

```
C:\ipsec>ipsec
IPSec Version 2.1.4 (c) 2001,2002 Marcus Mueller
Getting running Config ...
Microsoft's Windows XP identified
Host name is: (local_hostname)
No RAS connections found.
LAN IP address: (local_ip_address)
Setting up IPSec ...
Deactivating old policy...
Removing old policy...
Connection roadwarrior:
MyTunnel : (local_ip_address)
MyNet : (local_ip_address)/255.255.255.255
PartnerTunnel: (ip_of_remote_system)
PartnerNet : (ip_of_remote_system)/255.255.255.255
CA (ID) : C=FJ, ST=Fiji, L=Suva, O=SOPAC, OU=ICT, CN=SOPAC Root...
PFS : y
Auto : start
Auth.Mode : MD5
Rekeying : 3600S/50000K
Activating policy...
Connection roadwarrior-net:
MyTunnel : (local_ip_address)
MyNet : (local_ip_address)/255.255.255.255
PartnerTunnel: (ip_of_remote_system)
PartnerNet : (remote_subnet)/(remote_netmask)
CA (ID) : C=FJ, ST=Fiji, L=Suva, O=SOPAC, OU=ICT, CN=SOPAC Root...
PFS : y
Auto : start
Auth.Mode : MD5
Rekeying : 3600S/50000K
Activating policy...
C:\ipsec>
```

Now, ping your gateway host. It should say 'Negotiating IP Security' a few times, and then give you ping responses. Note that this may take a few tries; from a T1 hitting a VPN server on a cable modem, it usually takes 3−4 pings. Do the same for the internal network on the remote end, and you should be up!

Chapter 4. Global PKI

4.1. Current PKIs

At the moment you have the choice between a commercial PKI or your own PKI. The commercial PKI were created at the beginning to enable secure commerce over the Internet, basically securing HTTP. The pricing of certificates was calculated on a per host basis. The cost is more expensive than for a domain name because of the costs to identify the owner of the certificate (tracability), but also as a percentage into your e−commerce profits. Unfortunately this vision of a host basis has some major limitations. It is still acceptable to have a certificate to secure POP, IMAP, and other protocols, but when you need a certificate for each e−mail box on your network, costs start to skyrocket as well as the administrative burden to register all these certificates to the Certificate Authority and that every year. This problems exists too if you want to use certificates to authenticate clients in client/server applications (Web server, IPsec,..)

Why not have a certificate that can sign other certificates? At the moment the only option is to build your own Certificate Authority as described in this document. This allows flexible management of certificates but is limited to the people in your organisation, because people outside your organisation will have to load your root CA certificate to allow smooth operations.

The solution an unique PKI managed by a central authority in a similar format as DNS is managed. This is called a Global PKI.

4.2. The need for a Global PKI

In these days and age security on personnal computers has become important, such importance that Bill Gates stated that when Microsoft will have to choose between features and security, they will now choose security.

This reflections came from the increasing numbers of rogue people on the Internet. Anybody can send you anything, or trick you in installing anything on your computer. The solution is to identify everybody so when you have a problem you can at least blame someone. This is particulary true in the case of SPAM. It is often difficult to find the originator of an unsolicited e−mail and worse to be able to do something to stop this person. What many people have called for is tracability. If you receive some information which is not traceable through a certificate, you may decide to treat this information differently. This is the same concept as caller ID on telephone network. Certifcates offer this capaility for all applicationson the Internet, e−mails (S/MIME), Commerce transactions (HTTPS), Software install (Code Signing),... Unfortunately certificates are not widely used because they cost a lot if they have to be fully deployed. How many users on Yahoo mail, Hotmail, CA Online, can afford an e−mail certificate? There are some scheme to offer free e−mails certificates, but they can only certify that an e−mail address exists, they can trace back to a human or a body in the real world.

A global PKI is needed. All the protocols and standards exist, not need to reinvent the wheel. The IETF has all the mechanice worked out. An LDAP server can store the certificates, a DNS server can reference entry back to certificate stores, HTTP can deliver certificate to applications, S/MIME can secure e−mails,... The problem is now a policy problem or rather a profile problem: select which pieces of this standard should be used to cooperate into a global PKI. Which organisation should provide such service? What level of security/tracability will be achieved?... If one can answer these questions, it will be a step in the right direction and if users buy in, then problem solved...

I will keep updated this chapter as the work of the working group on PKI of the Internet Society progress. The Internet Society is also managing the .org Top Level Domain name, so they have a lot of capabilities at hand to solve this e−mail spamming problem.