Wszystko co powinieneś wiedzieć o tworzeniu certyfikatów ale nie chce Ci się poszukać w dokumentacji.

Co powinno znajdować się na Twoim dysku zanim zostaniesz "Certificate Authorities".

Podstawowym oprogramowaniem jest oczywiście openssl. W tym miejscu należy zachować czujność bo openssl **MUSI** być co najmniej w wersji 0.9.2b dzięki czemu ominie Cię część karkołomnych operacji przy pomocy pcks12 ktory także musisz posiadać w swoich zasobach dyskowych.
Jeśli masz już zainstalowane powyższe oprogramowanie możesz zacząć tworzyć certyfikaty.

Konfiguracja openssl.

Zakładam ze openssl jest zainstalowany standardowo czyli w **/usr/local/ssl**. Pierwszym krokiem jest przejrzenie i "dokonfigurowanie" **/usr/local/ssl/lib/openssl.cnf**. Mój domowy konfig wygląda następująco (kolorem czerwonym zaznaczylem opcje które raczej powinieneś zmienić) :
jeśli nie chce Ci się tego czytać to skocz na koniec konfiga

```
#
# 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

[ new_oids ]

# We can add new OIDs in here for use by 'ca' and 'req'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6

####################################################################
[ ca ]
default_ca    = CA_default          # The default ca section

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

dir           = ./demoCA            # Where everything is kept
certs         = $dir/certs          # Where the issued certs are kept
crl_dir       = $dir/crl            # Where the issued crl are kept
database      = $dir/index.txt      # database index file.
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
crl_extensions  = crl_ext             # Extensions to add to CRL
default_days    = 365                 # how long to certify for
default_crl_days= 30                  # how long before next CRL
default_md      = md5                 # which md to use.
preserve      = no                  # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
```

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

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

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

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

stateOrProvinceName             = State i Prowincja
stateOrProvinceName_default     = State-Prowincja domyslna
localityName                    = Locality Name (eg, city)
localityName_default            = Lodz

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = Nawza Organizacji

# 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  = Unit name domyslny

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

# This is typical also

keyUsage = nonRepudiation, digitalSignature, keyEncipherment

nsComment                       = "OpenSSL Generated Certificate"

# PKIX recommendations
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always
# 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_ca]

# Extensions for a typical CA

# It's a CA certificate
basicConstraints = CA:true

# 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

# Key usage: again this should really be critical.
keyUsage = cRLSign, keyCertSign

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

# 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

[crl_ext]

 # CRL extensions.
 # Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.

issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always,issuer:always
##
########## koniec pliku openssl.cnf

Jak widać zmiany są praktycznie kosmetyczne. Należy zwrócic jedynie uwagę na opcję default_bits w sekcji req. W momencie generowania certyfikatu CA powinna mieć ona wartość 1024 lub więcej, natomiast w trakcie tworzenia certyfikatów klienckich winno mieć się na uwadze wredną cechę produktów M$ dostępnych poza granicami USA. Nie są one w stanie zaimportować kluczy mających więcej niż 512 bitów. W takim przypadku default_bits należy zmniejszyć do tej wartości. Jeśli chodzi o Netscapa konieczność taka nie występuje, nawet gdy nie jest on patchowany przy pomocy Fortify. Jednakże klucz nie powinien być większy niż 1024 bity.

Generowanie certyfikatu CA

Pierwszyą czynnością jaką należy wykonać jest wygenerowanie certyfikatu CA czyli czegoś czym będą podpiswane certyfikaty udostępniane klientom. Uruchom rxvt lub coś innego i wykonaj polecenie:

adas:~# ***cd /usr/local/ssl/bin***
adas:/usr/local/ssl/bin# ***./CA.pl -newca***

CA certificate filename (or enter to create)

Making CA certificate ...
Using configuration from /usr/local/ssl/lib/openssl.cnf
Generating a 1024 bit RSA private key
..+++++
....+++++
writing new private key to './demoCA/private/cakey.pem'
Enter PEM pass phrase:
Verifying password - 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) [PL]:
State i Prowincja [Kraina Bezrobotnych Szwaczek]:
Locality Name (eg, city) [Lodz]:
Organization Name (eg, company) [Instytut Badan Czarow i Magii]:
Organizational Unit Name (eg, section) [Komorka d/s Egzorcyzmow i Opentan]:
Common Name (eg, YOUR name) []:Adam Hernik
Email Address []:adas@infocentrum.com

adas:/usr/local/ssl/bin#

Skrypt CA.pl uruchomiony poraz pierwszy tworzy w /usr/local/ssl/bin katalog o nazwie demoCA w którym znajduje się wygenerowany przed chwilą certyfikat publiczny **cacert.pem** (dołączany później do certyfikatów klienckich) oraz tajny zabezpieczony hasłem klucz **cakey.pem** którym będziesz podpisywał certyfikaty wydawane użytkownikom. Klucz i hasło oczywiście należy dobrze chronić i najlepiej jest gdy znajduje się na serwerze tylko w momencie generowania certyfikatu.
Ponowne uruchomienie CA.pl z parametrem -newca niszczy to co pracowicie stworzyłeś i generuje nowy klucz i certyfikat.

Tworzenie certyfikatu dla stunnela i innych serwerów

Zanim się do tego zabierzesz powinieneś lekko zmodyfikowac skrypt **CA.pl** oraz plik konfiguracyjny **openssl.cnf**. Skopiuj je odpowiednio do plików **/usr/local/ssl/bin/CAserv.pl** i **/usr/local/ssl/lib/openssl_serv.cnf**.
Generowane certyfikaty domyślnie zabezpieczone są hasłem, w takim przypadku w momencie startu stunnela zawsze będziesz pytany o haslo zabezpieczające, co skutecznie uniemożliwi automatyczne uruchamianie programu w czasie bootowania serwera, czy też przy próbie wystartowania go przez inetd. Należy poprawić **linie 40** i **41** skryptu

CAserv.pl z

linia 40:
$REQ="openssl req *$SSLEAY_CONFIG*";
na
$REQ="openssl req -nodes -config /usr/local/ssl/lib/openssl_serv.cnf";

linia 41:
$CA="openssl ca *$SSLEAY_CONFIG*";
na
$CA="openssl ca -config /usr/local/ssl/lib/openssl_serv.cnf";

Natomiast w pliku **/usr/local/ssl/lib/openssl_serv.cnf** nalezy w sekcji usr_cert "zahashować" linijkę nsCertType = client, email oraz "odhashować" linijkę nsCertType = server . Jeśli tego nie zrobisz klient nie będzie poprawnie rozpoznawał typu certyfikatu. A teraz kolej na wygenerowanie "requestu" posyłanego zazwyczaj do CA. Będąc w katalogu /usr/local/ssl/bin wykonaj:

*adas:/usr/local/ssl/bin# **./CAserv.pl -newreq***
Using configuration from /usr/local/ssl/lib/openssl_serv.cnf
Generating a 1024 bit RSA private key
.....+++++
.....+++++
writing new private key to 'newreq.pem'

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) [PL]:
State i Prowincja [Kraina Bezrobotnych Szwaczek]:Kraina latajacych scyzorykow
Locality Name (eg, city) [Lodz]:Sielpia
Organization Name (eg, company) [Instytut Badan Czarow i Magii]:Bar Sloneczko
Organizational Unit Name (eg, section) [Komorka d/s Egzorcyzmow i Opentan]:Kuflownia
Common Name (eg, YOUR name) []:adas.pl
Email Address []:adas@adas.pl

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Request (and private key) is in newreq.pem
adas:/usr/local/ssl/bin#

Polem o którym warto wspomnieć jest "Common Name" (zaznaczone na czerwono). W trakcie generowania requestu należy w tym miejscu wpisać **FQDN serwera** na którym będzie on używany. W przeciwnym wypadku w chwili połączenia klient będzie twierdził, że certyfikat jakim przedstawia się serwer nie należy do niego. Unikniemy w ten sposób niepotrzebnego klikania. Kolejną czynnością jest podpisanie wygenerowanego requestu. W katalogu /usr/local/ssl/bin wykonaj polecenie:

*adas:/usr/local/ssl/bin# **./CAserv.pl -sign***
Using configuration from /usr/local/ssl/lib/openssl.cnf
Enter PEM pass phrase:
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName :PRINTABLE:'PL'
stateOrProvinceName :PRINTABLE:'Kraina latajacych scyzorykow'
localityName :PRINTABLE:'Sielpia'
organizationName :PRINTABLE:'Bar Sloneczko'
organizationalUnitName:PRINTABLE:'Kuflownia'
commonName :PRINTABLE:'adas.pl'
emailAddress :IA5STRING:'adas@adas.pl'
Certificate is to be certified until Mar 26 21:06:13 2000 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

Signed certificate is in newcert.pem
adas:/usr/local/ssl/bin#

W trakcie podpisywania będziesz pytany o hasło zabezpieczające klucz prywatny CA (zaznaczone na zielono). Po tej operacji powinieneś w katalogu /usr/local/ssl/bin otrzymać 2 pliki: **newcert.pem** oraz **newreq.pem**. Zanim zaczniesz ich używać musisz wykonać jeszcze jedną operacje, a mianowicie złorzyć wszystko do kupy. Wykonujesz: **cat newcert.pem newreq.pem > httpds.pem** a następnie poddajesz tak powstały certyfikat edycji. Należy z pliku httpds.pem należy usunąć wszystkie niepotrzebne informacje tak by pozostał jedynie certyfikat oraz klucz prywatny. Po tej operacji plik httpds.pem powinien wyglądać mniej więcej tak:

issuer :/C=PL/ST=Kraina Bezrobotnych Szwaczek/L=Lodz/O=Instytut Badan Czarow i Magii/OU=Komorka d/s Egzorcyzmow i opentan/CN=Adam Hernik/Email=adas@infocentrum.com
subject:/C=PL/ST=Kraina latajacych scyzorykow/L=Sielpia/O=Bar Sloneczko/OU=Kuflownia/CN=adas.pl/Email=adas@adas.pl
-----BEGIN CERTIFICATE-----
Tu są magiczne dane
-----END CERTIFICATE-----

-----BEGIN RSA PRIVATE KEY-----
I tu też są magiczne dane
-----END RSA PRIVATE KEY-----

Spreparowany w ten sposób plik umieszczamy w katalogu /usr/local/ssl/certs i zajmujemy się generowaniem dwu certyfikatów klienckich.

Generowanie i importowanie certyfikatów klienckich do Netscape Communikatora.

Generalnie są dwie metody tworzenia i importowania certyfikatów klienckich do Netscapa
Sposób pierwszy:
Przy pomocy komendy **CA.pl -newreq** wygeneruj request a następnie przy pomocy **CA.pl -sign** podpisz go. Pytanie o *challenge password* zignoruj. Kolejną czynnością jest scalenie i podczyszczenie certyfikatu. W przypadku certyfikatu klienta ważne jest podanie **prawidłowego adresu email !** Bez tego nie będzie można podpisywać i szyfrować listów. Stwórz dwa certyfikaty. Będą one potrzebne do wyjaśnienia działania opcji -v 3 programu stunnel. Zakładam że pierwszy certyfikat należy do Jana Kowalskiego jan@ibczim.pl zachowany w pliku jan.pem a drugi do Genowefy Pigwy pigwa@scyzoryki.pl znajdującym się w pliku pigwa.pem. Przed zaimportowaniem plików do Netscpea należy przekonwertować je z formatu PEM do PCKS12. Wykonuje się to przy pomocy wspomnianego na początku programu **pcks12**. Aby przekonwertować certyfikat Jan Kowalskiego, w katalogu w ktorym znajduje się plik jan.pem wykonaj:

pkcs12 -export -name "Jan Kowalski jan@ibczim.pl" -in jan.pem -out jan.p12 -certfile /usr/local/ssl/bin/demoCA/cacert.pem

(jest to jedna linia !!!)
w wyniku czego powstanie plik jan.p12 który można zaimportować do Netscapea. Bardzo ważną opcją jest ***-certfile /usr/local/ssl/bin/demoCA/cacert.pem***. Bez niej nie będzie można w prawidłowy sposób podpisywać listów. Przełącznik -certfile powoduje dołączenie publicznego certyfikatu CA do certyfikatu klienta dzięki czemu Netscape jest wstanie "wyekstrachować" certyfikat CA i dodać go do wewnętrznej bazy CA. Wykonaj powyższą operację także dla pigwy. Samo zaimportowanie certyfikatu jest bardzo proste wykonuje się to klikając w Netscape na

Security-> Yours -> Import a Certificate

Po zaimportowaniu należy w **Security -> Signers** zaznaczyć nasz CA certyfikat a następnie kliknąć na przycisku Edit oraz "zaczekować" opcje:

Accept this Certificate Authority for Certifying network sites
Accept this Certificate Authority for Certifying e-mail users

Od tej pory nasz certyfikat będzie traktowany na równi z innymi, komercyjnymi.

Sposób drugi:
Polega on na wygenerowaniu i imporcie certyfikatu poprzez strone www. Wraz z stunnelem dostarczane są przkładowe strony (dwie) i skrypty (dwa). Skrypty należy raczej traktować jako wzorzec i każdy powinien napisać swoje, bardziej bezpieczne. Pierwszym krokiem jest import certyfikatu CA. Używa się do tego strony **importCA.html** oraz skryptu **importCA.sh**. Sam skrypt wygląda tak:

#!/bin/bash

echo "Content-type: application/x-x509-ca-cert"
echo
*cat /var/lib/httpds/cgi-bin/**cacert.pem***

cacert.pem jest to oczywiście certyfikat publiczny CA znajdujący się w katalogu /usr/local/ssl/bin/demoCA

który należy przekopiować do katalogu cgi-bin serwera httpd oraz nadać mu odpowiednie prawa dostępu.
Po zaimportowaniu certyfikatu CA należy w Security->Signers zaznaczyć do jakich celów będziemy uznawli go za wiarygodny. Do generowania certyfikatu klienta wykorzystamy pozostałą strone i skrypt. Zanim do tego dojdzie należy "dokonfigurować" skrypt i stworzyć potrzebne katalogi. W /tmp (lub w innym miejscu) nalezy stworzyć katalog ssl a następnie przekopiować do niego katalog **/usr/local/bin/demoCA** oraz plik **openssl.cnf**.
Jako że skrypty domyślnie uruchamiane są z prawami użytkownika nobody należy uczynić go właścicielem katalogu /tmp/ssl i całej jego zawartości. Kolejną czynnością jest wygenerowanie pliku **.rnd**. W Linuxie robimy to tak:
cat /dev/random > /tmp/ssl/.rnd
czekamy chwilkę tak by plik .rnd miał wielkość około 1024 B po czym właścicielem pliku robimy użytkownika nobody.
Teraz trzeba przekonfigurować plik /tmp/ssl/openssl.cnf

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

RANDFILE              = /tmp/ssl/.rnd
#oid_file             = /tmp/ssl/.oid
oid_section           = new_oids

[ 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           = /tmp/ssl/demoCA             # Where everything is kept
certs         = $dir/certs          # Where the issued certs are kept
crl_dir       = $dir/crl            # Where the issued crl are kept
database      = $dir/index.txt      # database index file.
new_certs_dir = $dir/newcerts       # default place for new certs.
```

Należy zmienić opcje zaznaczone na czerwono. Ostatnią czynnością jest sprawdzenie i ewentualne poprawienie strony ca.html i skryptu ca.pl. W pliku ca.html nalezy wpisać poprawną nazwę serwera na którym znajduje się skrypt ca.pl czyli linijkę **<FORM ACTION="http://localhost/cgi-bin/ca.pl" METHOD=POST>**. W ca.pl należy skontrolować poprawność podanych ścieżek oraz wpisać hasło jakim zabezpieczony jest klucz prywatny CA (zmienna $certpass zaznaczona na czerwono).

```
#!/usr/bin/perl
#ca.pl

$config   = "/tmp/ssl/openssl.cnf";
$capath   = "/usr/local/ssl/bin/openssl ca";
$certpass = "tu_jest_haslo";
$tempca   = "/tmp/ssl/cli".rand 10000;
$tempout  = "/tmp/ssl/certtmp".rand 10000;
$caout    = "/tmp/ssl/certwynik.txt";
$CAcert   = "/tmp/ssl/demoCA/cacert.pem";
...
```

Po umieszczeniu tak przygotowanych stron i skryptów na serwerze będzie można generować certyfikaty dla klientów.

Wady i zalety obydwu sposobów generowania i instalowania certyfikatów.

Jak wynika z powyższego opisu bezpieczniejszym i polecanym przeze mnie jest sposób pierwszy. Jego poważną wadą jest fakt że człowiek generujący certyfikaty znajduje się w posiadaniu klucza prywatnego osoby występującej o certyfikat. Oczywiście uczciwy CA powinien skasować go, zaraz po utworzeniu. W takim wypadku metoda pierwsza spełnia wszelkie wymogi. Sposób drugi prócz samych wad ma jedną acz ogromną zaletę. Mianowicie klucz prywatny klienta nigdy nie opuszcza jego komputera. Do wad można zaliczyć fakt że hasło zabezpieczające klucz prywatny CA znajduje się na serwerze i to w dodatku w żaden sposób nie chronione. Kolejną wadą jest generowanie kompletnych certyfikatów przez strone www, co może grozić wykradzeniem klucza prywatnego. Rozwiązaniem może być składowanie

requestów w bazie danych a nastpnie ręczna ich obróbka przez administratora. Reasumując, sposób drugi należy potraktować jako demonstracje metody którą można przećwiczyć przed napisaniem porządnych skryptów.

Tajemniczy przełącznik -v 3 w stunnelu

Stunnel posiada trzy tryby weryfikacji klienta.

Pierwszy opcja **-v 1** oznacza że należy spróbować zweryfikować osobę nawiązującą połączenie czyli uzyskać jej ceryfikat. Jeśli operacja ta się nie powiedzie, mimo wszystko dostęp do serwera będzie zapewniony.

Przełącznik **-v 2** nakazuje stunnelowi zweryfikować klienta. Jeśli użytkownik nie posiada certyfikatu lub certyfikat jest nieważny, niewłaściwy czy też nie posiadamy certyfikatu CA którym podpisany jest certyfikat klienta

(straszny jest ten język polski) nawiązanie połączenia z serwerem będzie niemożliwe. I wreszcie opcja **-v 3** nakazująca stunnelowi zweryfikować klienta a także poszukać jego certyfikatu w naszej lokalnej bazie.
Dzieki opcji -v 3 możemy stworzyć bardzo selektywny dostęp do usług oferowanych przez serwer, unikając generowania dużych ilości certyfikatów. Uwaga ogólna: do poprawnej weryfikacji klienta KONIECZNE jest posiadanie certyfikatu CA którym podpisany jest sprawdzany certyfikat. Bez tego stunnel nie jest wstanie przeprowadzić poprawnej autoryzacji klienta. Próba taka kończy się błędami "**VERIFY ERROR: self signed certificate for .....**" oraz "**SSL_accept: error:140890B1:SSL routines: SSL3_GET_CLIENT_CERTIFICATE:no certificate returned**". A teraz przykład praktyczny: chcemy aby do https będącym na **porcie 444** miały dostęp wszystkie osoby mające certyfikaty natomiast do do https na **porcie 445** dostęp miał tylko Jan Kowalski. Pierwszą czynnością jaką należy wykonać jest skopiowanie certyfikatu CA do katalogu **/usr/local/ssl/certs** (default cert area), następnie w tym katalogu należy utworzyć podkatalog o nazwie **mytrusted**, poczym skopiować do niego certyfikat klienta czyli jan.pem. **Uwaga**: z pliku jan.pem **MUSISZ** usunąć klucz prywatny !!! Czyli to co się znajduje między

-----BEGIN RSA PRIVATE KEY-----
.....
-----END RSA PRIVATE KEY-----

łącznie z powyższymi liniami. Następnie w katalogach **/usr/local/ssl/certs** i **/usr/local/ssl/certs/mytrusted** należy wykonać polecenie
/usr/local/ssl/bin/c_rehash ./
Teraz kolej na uruchomienie stunnela:
stunnel -d 444 -r 80 -v 2
oraz
stunnel -d 445 -r 80 -v 3
Netscapem należy połączyć sie z https://localhost:444/ a po pytaniu o certyfikat przedstawić certyfikat należący do pigwy. Dostęp do serwera będzie zapewniony. Czynnośc tę należy powtórzyć przedstawiając się za drugim razem certyfikatem Jana Kowalskiego. Połączenie także będzie zrealizowane. W przypadku https://localhost:445/ wejście na serwer będzie zapewnione tylko po wylegitymowaniu się certyfikatem Jana Kowalskiego. Po kazdej zmianie w katalogu /usr/local/ssl/certs/mytrusted należy wykonać komendę c_rehash ./ i zrestartować stunnela.