

# Wprowadzenie do OpenSSL

Witold Paluszyński

Katedra Cybernetyki i Robotyki

Politechnika Wrocławska

<http://www.kcir.pwr.edu.pl/~witold/>

2013



Ten utwór jest dostępny na licencji  
**Creative Commons Uznanie autorstwa-  
Na tych samych warunkach 3.0 Unported**

Utwór udostępniany na licencji Creative Commons: uznanie autorstwa, na tych samych warunkach. Udziela się zezwolenia do kopiowania, rozpowszechniania i/lub modyfikacji treści utworu zgodnie z zasadami w/w licencji opublikowanej przez Creative Commons. Licencja wymaga podania oryginalnego autora utworu, a dystrybucja materiałów pochodnych może odbywać się tylko na tych samych warunkach (nie można zastrzec, w jakikolwiek sposób ograniczyć, ani rozszerzyć praw do nich).



# SSL i TLS

SSL (Secure Sockets Layer) jest standardem bezpiecznej komunikacji w Internecie. Polega na wbudowaniu technologii szyfrowania danych opartej na infrastrukturze klucza publicznego (Public Key Infrastructure, PKI) w protokół komunikacyjny. Dane są szyfrowane kluczem odbiorcy przed wysłaniem ich. Dzięki temu są zabezpieczone przed możliwością ich odczytania przez nieuprawnioną stronę trzecią, jak również są odporne na manipulację. Jednocześnie system dystrybucji kluczy rozwiązuje problem ich autentyczności.

Bezpieczne połączenia oparte na SSL mogą być stosowane do każdego rodzaju sieciowego protokołu komunikacyjnego, np. HTTP, POP3, FTP, telnet, itd.

SSL nie jest nową technologią. Bieżąca wersja protokołu 3.0 istnieje od 1996. Planowane jest zastąpienie go przez nowszy protokół TLS (Transport Layer Security), który jest podobny ale niekompatybilny. SSL i TLS są najpowszechniej wspieranymi przez serwery WWW szyfrowanymi protokołami (99.8% serwerów wspiera wersję 3.0 SSL, 99.4% wspiera wersję 1.0 TLS).<sup>1</sup>

---

<sup>1</sup>Stan z kwietnia 2013.

# OpenSSL

OpenSSL jest przenośną, wieloplatformową implementacją protokołów SSL i TLS, dostępną na zasadach *open source*. Zasadniczo OpenSSL ma postać biblioteki ANSI C, która implementuje podstawowe operacje kryptograficzne. Poza funkcjami niezbędnymi do szyfrowania sieciowej warstwy transportu, zawiera również funkcje szyfrowania symetrycznego (dla plików), podpisy cyfrowe, kryptograficzne funkcje skrótu, generatory liczb losowych itp.

OpenSSL jest więcej niż tylko API, to także program użytkownika z interfejsem wiersza polecenia. Program pozwala na to samo, co API, i dodatkowo, pozwala sprawdzać serwery i klientów SSL.

W tym wykładzie przedstawione są podstawowe możliwości OpenSSL dostępne z narzędzia terminalowego.

Istnieje inna implementacja typu open-source protokołów SSL/TLS — GnuTLS. Zasadnicza różnica między tymi pakietami jest w typie licencji darmowej. Jednak GnuTLS nie posiada narzędzia terminalowego, i z punktu widzenia użytkownika systemów uniksowych jest mało interesujący.

# Funkcje programu openssl

Program openssl umożliwia następujące operacje:

- Tworzenie i zarządzanie kluczami prywatnymi, publicznymi i ich parametrami
- Operacje kryptograficzne z kluczem publicznym
- Tworzenie certyfikatów X.509, CSR oraz CRL
- Obliczanie skrótów wiadomości
- Szyfrowanie i deszyfrowanie różnymi szyframi
- Testowanie klientów i serwerów SSL/TLS
- Przetwarzanie poczty podpisanej lub zaszyfrowanej S/MIME
- Żądania znaczników czasowych, generowanie i weryfikacja

# Podstawowe wywołania openssl

Program openssl wywołuje się z wektorem argumentów definiującym funkcję:

```
# sprawdzenie zainstalowanej wersji openssl
openssl version
# z obszernymi informacjami
openssl version -a

# lista poleceń openssl: jakiegokolwiek nieznanego polecenie, np.
openssl help
# podobny trik działa dla indywidualnych poleceń openssl, np.
openssl dgst -h

# kompleksowe testy wydajności operacji szyfrowania systemu
openssl speed
# testy wydajności ograniczone do konkretnego algorytmu
openssl speed rsa
# testy wydajności z uwzględnieniem wieloprosesorowości
openssl speed rsa -multi 2
```

Można również wejść w tryb dialogowy openssl i pisać jego polecenia. Jednak brak wtedy możliwości readline — edycji poleceniami Emacs, historii, itp.

# Dostępne algorytmy szyfrowania openssl

```
openssl list-cipher-commands
```

```
# lista dostępnych algorytmów z pełną informacją
```

```
openssl ciphers -v
```

```
# lista tylko szyfrów wersji TLSv1
```

```
openssl ciphers -v -tls1
```

```
# lista szyfrów "mocnych" (klucze powyżej 128 bitów)
```

```
openssl ciphers -v 'HIGH'
```

```
# lista szyfrów "mocnych" z algorytmem AES
```

```
openssl ciphers -v 'AES+HIGH'
```

# Testowanie zdalnego serwera WWW

Openssl pozwala wykonywać wiele różnych testów zdalnych serwerów HTTPS:

```
# testowanie nawiązywania polaczen ze zdalnym serwerem WWW przez 30s  
openssl s_time -connect remote.host:443
```

```
# test.polaczen i sciagania strony przez 10 sekund, tworz.nowej sesji  
openssl s_time -connect remote.host:443 -www /index.html -time 10 -new
```

Jeśli nie mamy do dyspozycji zdalnego serwera HTTPS, który możnaby wykorzystać do testów, openssl pozwala „postawić” minimalny serwer na wybranym porcie. Serwer serwuje pliki z lokalnego katalogu, w którym został uruchomiony:

```
# uruchomienie serwera HTTPS na porcie 10443 z certyfikatem mycert.pem  
openssl s_server -accept 10443 -cert mycert.pem -WWW
```

```
# wygenerowanie certyfikatu serwera, zadaje duzo szczegolowych pytan  
openssl req \  
  -x509 -nodes -days 365 \  
  -newkey rsa:1024 -keyout mycert.pem -out mycert.pem
```



# Symetryczne szyfrowanie plików

Szyfrowanie symetryczne jest pomocniczą funkcją `openssl`. W tej roli `openssl` jest mniej wygodny w użyciu niż np. GnuPG. Przy deszyfrowaniu wymaga znajomości, oprócz hasła, algorytmu użytego do szyfrowania.

```
# szyfruj file.txt do file.enc algorytmem AES-256-CBC,  
openssl enc -aes-256-cbc -salt -in file.txt -out file.enc  
# alternatywna forma tego samego, z kodowaniem tekstowym base64  
openssl aes-256-cbc -a -salt -in file.txt -out file.ascii  
  
# deszyfruj plik binarny na stdout  
openssl enc -d -aes-256-cbc -in file.enc  
# deszyfruj plik zakodowany tekstowo base64 na stdout  
openssl enc -d -aes-256-cbc -a -in file.ascii  
  
# hasło szyfrowania można podać w wierszu polecenia  
openssl enc -aes-256-cbc -salt -in file.txt -out file.enc \  
-pass pass:Kathy123  
# można również hasło zapisać na pliku  
openssl enc -aes-256-cbc -salt -in file.txt -out file.enc \  
-pass file:~/passwords/mypassword.txt
```

```
# koduj tresc pliku base64 (bez szyfrowania), zapisz na drugim pliku
openssl enc -base64 -in file.txt -out file.ascii
# koduj base64 "w locie", wyslij wynik na wyjscie, uwaga na NEWLINE
printf "jakis napis" | openssl enc -base64
# dekodowanie BASE64, ponownie uwaga na NEWLINE
printf "amFraXMgmbmFwaXM=\n" | openssl enc -base64 -d
```

# Skróty kryptograficzne i podpisy cyfrowe

Skróty kryptograficzne (*file hash* lub *message digest*) pełnią rolę sygnatur dużych plików danych. Na przykład, zamiast porównywać pliki każdy z każdym, można obliczyć ich skróty i szybko je porównać. Jeszcze ważniejszą rolę skróty pełnią przy cyfrowym podpisywaniu przesyłanych danych. Zamiast podpisywać cały plik, co jest równoważne z jego zaszyfrowaniem, można podpisać jego skrót.

Zadaniem skrótu kryptograficznego jest być „prawie unikalnym”. To znaczy, znalezienie innego pliku z tym samym skrótem co dany plik musi być bardzo trudne. W praktyce, podpisanie skrótu jest mniej bezpieczne niż całego pliku.

```
# skrot MD5
openssl dgst -md5 filename
# skrot SHA1
openssl dgst -sha1 filename

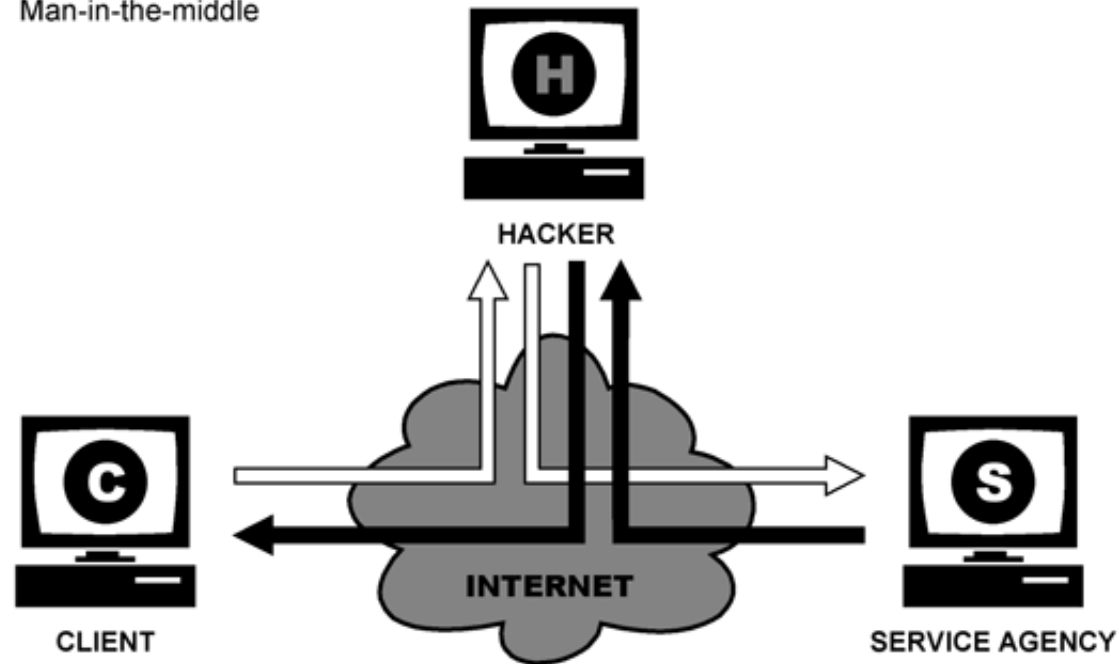
# wygenerowanie podpisu skrotu sha1 kluczem prywatnym
openssl dgst -sha1 -sign mykey.pem -out foo.tar.gz.sha1 foo.tar.gz

# weryfikacja sygnatury z kluczem publicznym nadawcy
openssl dgst -sha1 -verify pubkey.pem -signature foo.tar.gz.sha1 \
foo.tar.gz
```



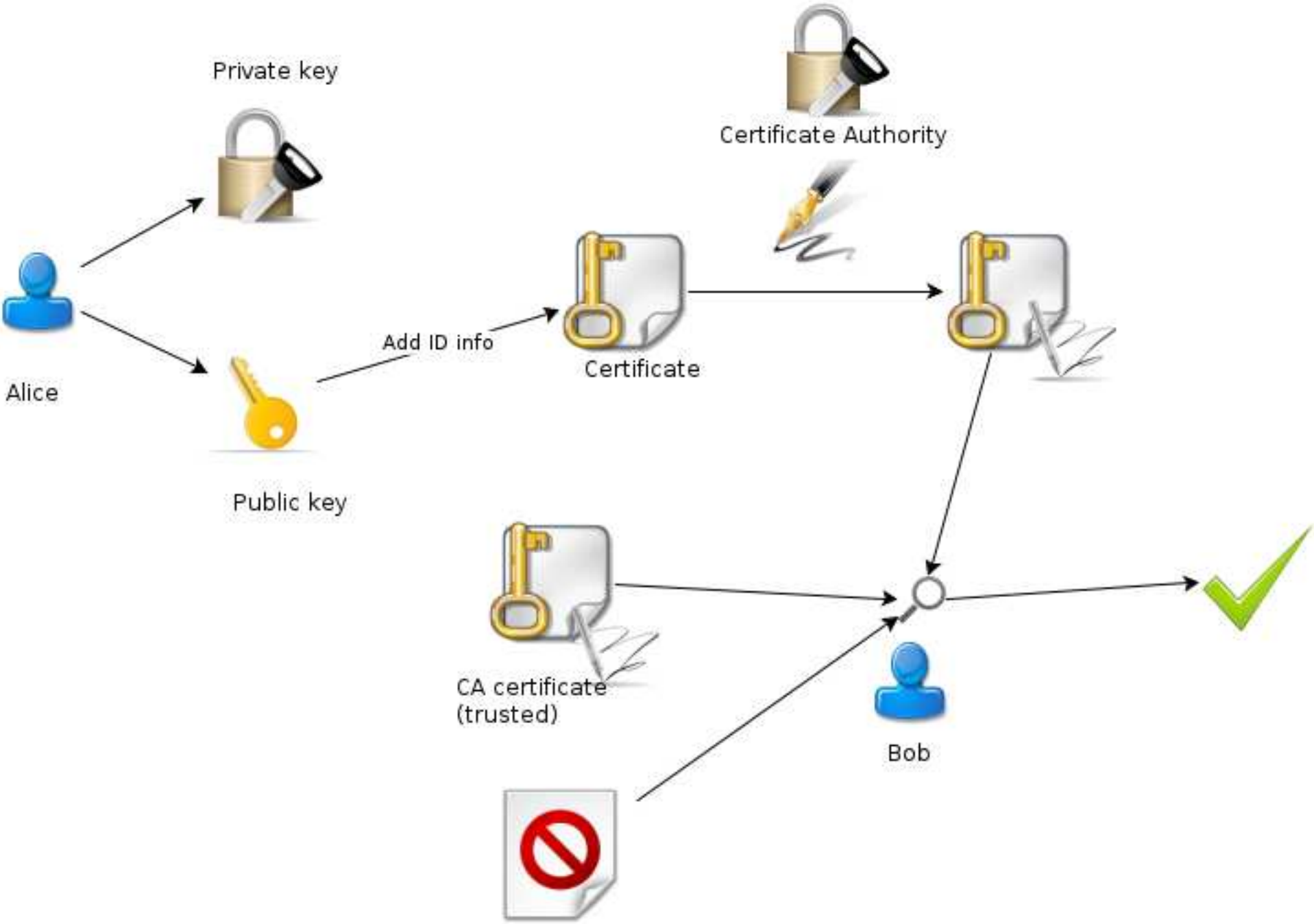
# Infrastruktura klucza publicznego

Man-in-the-middle



- PKCS#1 — RSA
- PKCS#7 — S/MIME
- PKCS#8 — klucz prywatny
- PKCS#10 — żądanie certyfikatu
- PKCS#11 — API sprzętu kryptograficznego
- PKCS#12 — klucz prywatny z certyfikatami
- PKCS#15 — tokeny kryptograficzne

# Praca z kluczami PKI



# Generowanie kluczy

Prywatne klucze RSA (PKCS#8) można wygenerować poniższymi poleceniami:

```
# wygeneruj 1024-bitowy klucz prywatny RSA, zapisz na pliku mykey.pem  
openssl genrsa -out mykey.pem 1024
```

```
# jak wyzej, dodatkowo zakoduj plik haslem  
openssl genrsa -des3 -out mykey.pem 1024
```

```
# wyświetlenie zawartosci klucza prywatnego w postaci tekstowej  
openssl rsa -in mykey.pem -text -noout
```

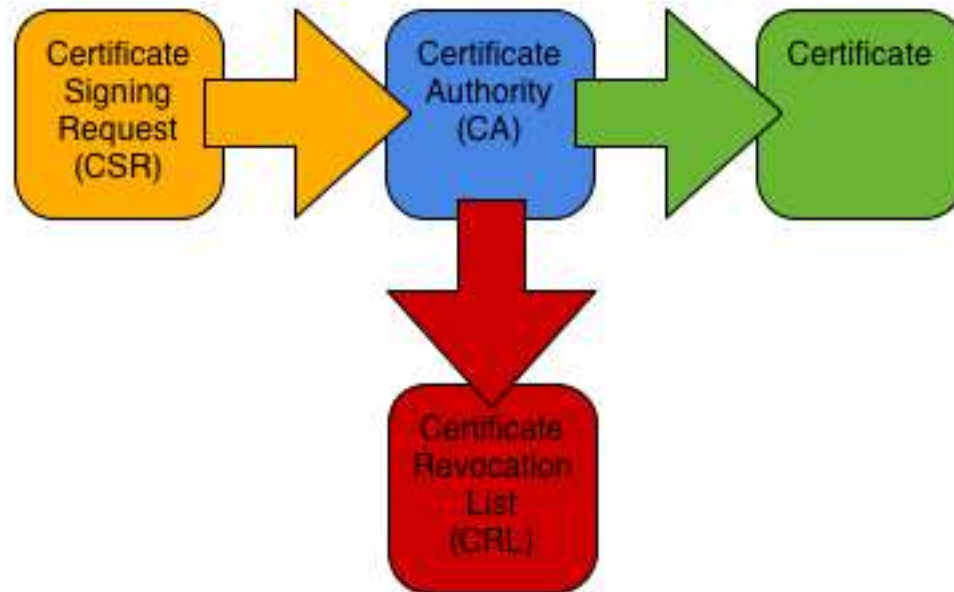
```
# wygeneruj na stdout klucz publiczny do posiadanego klucza prywatnego  
openssl rsa -in mykey.pem -pubout
```

Przy generowaniu kluczy podstawową kwestią (oprócz typu i długości kluczy), jest czy mają być chronione hasłem. Poczucie bezpieczeństwa wymagałoby, aby były. Oznacza to, że przy każdym dostępie do klucza prywatnego trzeba wpisywać to hasło. Dlatego czasami jest racjonalne zrezygnowanie z hasła i zabezpieczenie pliku z kluczem prywatnym tylko prawami dostępu.

# Generowanie żądania certyfikatu

W systemie PKI bardziej niż bezpośredni klucz publiczny potrzebny jest jednak certyfikat podpisany przez CA (Certification Authority). Należy więc wygenerować dodatkowy plik zawierający żądanie takiego certyfikatu (PKCS#10), i wysłać go do centrum certyfikacji (oraz zapłacić należność):

```
# generowanie zadania certyfikatu, seria pytan
openssl req -new -key mykey.pem -out mykey_csr.pem
# mozna rowniez wygenerowac klucz razem z zadaniem certyfikatu
openssl req -new -newkey rsa:1024 -nodes -keyout mykey.pem \
-out mykey_csr.pem
```





# Sprawdzanie żądania certyfikatu

Żądanie certyfikatu (PKCS#10) i otrzymany w jego wyniku podpisany certyfikat są elementami oficjalnej działalności instytucji CA. Do obowiązków CA należy dokładne sprawdzenie tożsamości wnioskodawcy, jako że podpisany certyfikat będzie służył do późniejszego potwierdzania jego tożsamości podpisami cyfrowymi. Istotne jest więc, aby dane w żądaniu PKCS#10 były w 100% poprawne. Można je odczytać, jak również sprawdzić zgodność podpisu zawartego w żądaniu z kluczem prywatnym.

```
# sprawdź informacje zawarte w zadaniu certyfikatu  
openssl req -in mykey_csr.pem -noout -text
```

```
# sprawdź zgodność podpisu zawartego w zadaniu certyfikatu  
openssl req -in mykey_csr.pem -noout -verify -key mykey.pem
```

# Zróbmy sobie CA

Teraz należałoby zaczekać na otrzymanie podpisanego certyfikatu z CA. Aby skrócić czas oczekiwania, możemy zbudować własne CA, i samodzielnie podpisać nasz certyfikat. Ma to sens często w sytuacjach praktycznych, gdy własne, prywatne CA jest wystarczające do funkcjonowania budowanego systemu.

```
CA_DIR=./demoCA
```

```
echo "Setting up the directories"
```

```
mkdir -p ${CA_DIR} ${CA_DIR}/certs ${CA_DIR}/private \
        ${CA_DIR}/newcerts ${CA_DIR}/crls
```

```
chmod -R 700 ${CA_DIR}
```

```
cd ${CA_DIR}
```

```
echo 1000 > serial
```

```
touch index.txt
```

```
echo "Creating the CA, no password ..."
```

```
openssl req -nodes -new -x509 -days 3650 -keyout private/cakey.pem \
    -out cacert.pem -subj "/CN=witoldp@pwr.wroc.pl/O=Wroclaw Univ. of \
    Technology/OU=I6/C=PL/ST=Dolnoslaskie/L=Wroclaw"
```

# Podpisywanie certyfikatów

Teraz przed nami najlepsze, czyli możemy podpisywać certyfikaty przysłane w żądaniach:

```
# aby obejrzec otrzymane zadanie podpisania certyfikatu  
openssl req -noout -text -in mykey_csr.pem
```

```
# podpisanie certyfikatu  
openssl ca -in mykey_csr.pem -out wpcert.pem
```

Podpisany certyfikat zapisany w podanym pliku, jak również w podkatalogu newcerts CA.

# Posługiwanie się certyfikatem

Klient żądał podpisania certyfikatu, żeby móc posługiwać się nim dla uwiarygodnienia swoich transakcji. Do jakich transakcji może go wykorzystywać:

- szyfrowanie poczty standardem S/MIME

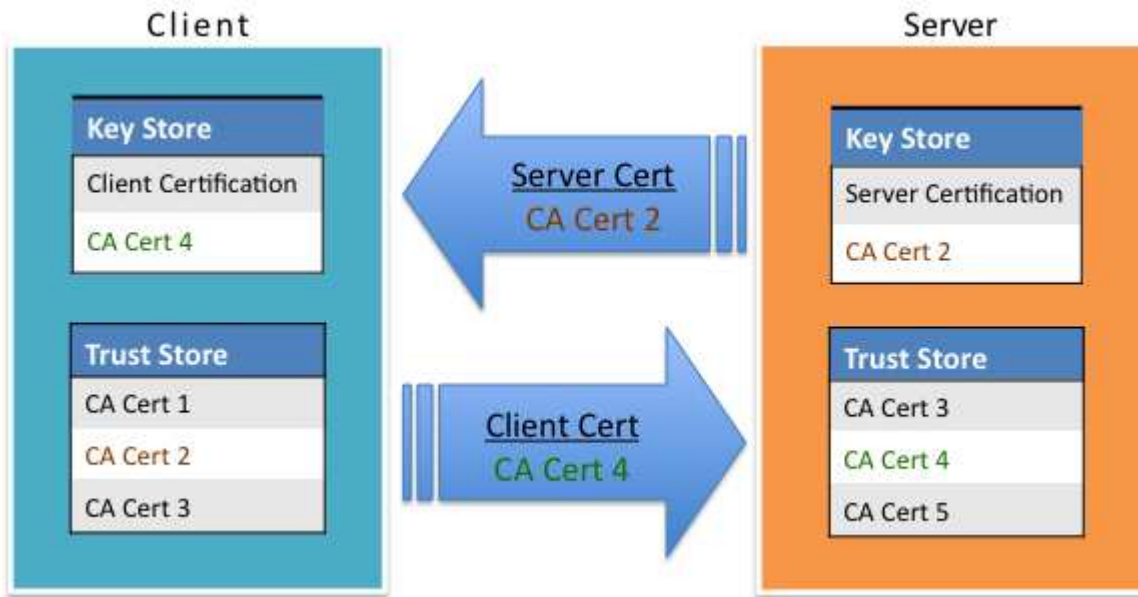
```
openssl smime -encrypt -in /etc/hosts -from lola@pwr.wroc.pl \  
-to tyciu@stud.ict.pwr.wroc.pl -subject sub rec_cert.pem
```

- podpisy cyfrowe w poczcie standardem S/MIME

```
openssl smime -sign -in in.txt -text -signer mycert.pem \  
-from steve@openssl.org -to someone@somewhere \  
-subject "Signed message"
```

- potwierdzanie wiarygodności serwera WWW:

```
cp userkey.pem /opt/csw/apache/conf/ssl.key/  
cp usercert.pem /opt/csw/apache/conf/ssl.crt/
```



## Przydatne linki

Paul Heinlein, OpenSSL Command-Line HOWTO

<http://www.madboa.com/geek/openssl/>

J.K. Harris, Understanding SSL/TLS

[http://computing.ece.vt.edu/~jkh/Understanding\\_SSL\\_TLS.pdf](http://computing.ece.vt.edu/~jkh/Understanding_SSL_TLS.pdf)

Philippe Camacho, An Introduction to the OpenSSL command line tool

[http://users.dcc.uchile.cl/~pcamacho/tutorial/crypto/openssl/openssl\\_intro.html](http://users.dcc.uchile.cl/~pcamacho/tutorial/crypto/openssl/openssl_intro.html)

Sun/Oracle documents, Introduction to SSL

<http://docs.oracle.com/cd/E19957-01/816-6156-10/>