

Przykłady algorytmów kryptograficznych

Algorytmy symetryczne

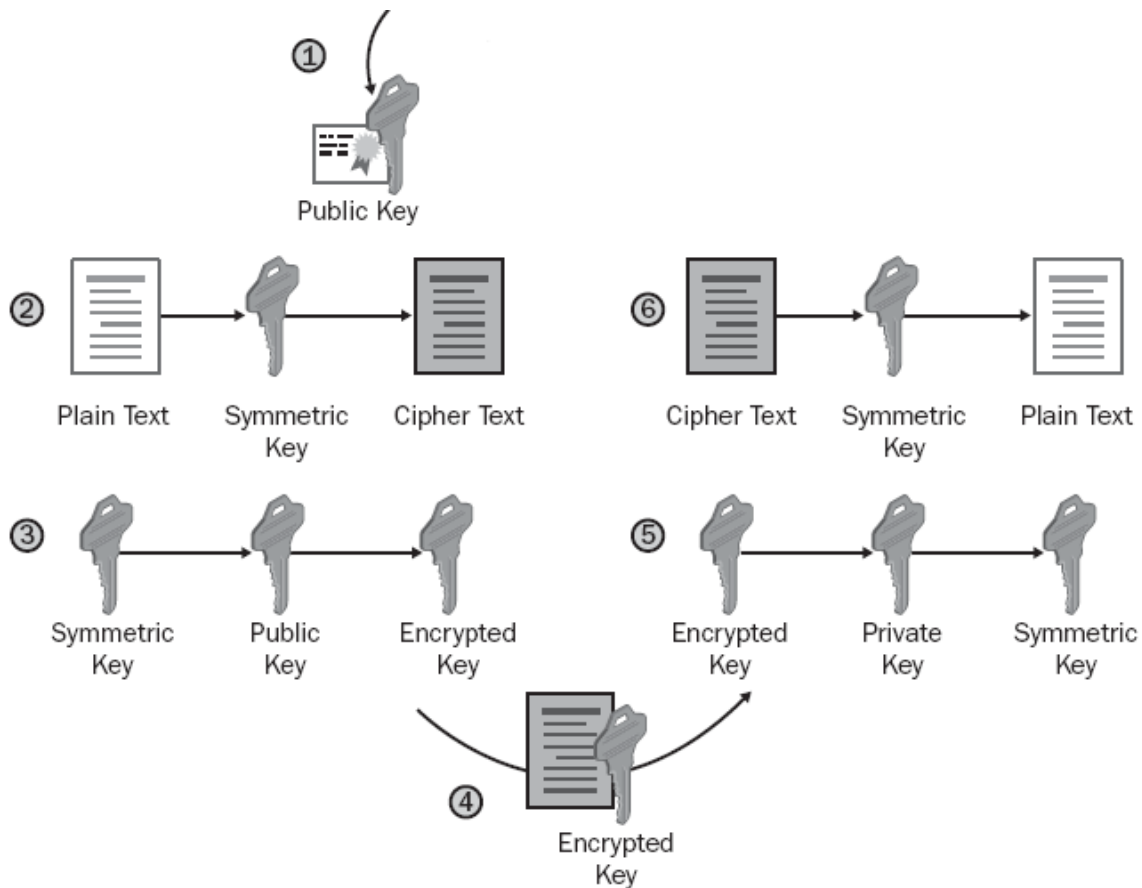
- Data Encryption Standard (DES) – algorytm blokowy używający klucza 56 bitowego,
- Data Encryption Standard XORed (DESX) – przed zaszyfrowaniem z użyciem DES, dane są XORowane z dodatkowym ciągiem 64bitów, co nieco poprawia poziom bezpieczeństwa,
- Rivest's Cipher version 2 (RC2 – 40bit) – algorytm blokowy, pracujący na blokach długości 64 bitów; wykorzystuje dodatkowo ciąg 40 bitów (tzw. salt) dodawany do klucza szyfrującego przed jego wykorzystaniem,
- RC2 (128 bit) – odmiana RC2 gdzie salt ma długość 88 bitów,
- RC4 – algorytm strumieniowy, wykorzystujący klucze różnej długości, często używany np. w protokole SSL,
- Triple DES (3DES) – Odmiana DES, gdzie szyfrowanie odbywa się 3 razy z różnymi kluczami 56 bitowymi: najpierw dane są szyfrowane kluczem A, potem rozszyfrowywane kluczem B, a na koniec szyfrowane kluczem C. Razem daje to odpowiednik klucza 168 bitowego.
- Advanced Encryption Key (AES) – opracowany jako następca DES algorytm blokowy, wykorzystuje klucze 128, 196 i 256 bitowe.

Algorytmy asymetryczne

- Diffie-Hellman key agreement (DH) – nie służy do szyfrowania, lecz pozwala 2 stronom na bezpieczne uzgodnienie tajnego materiału kryptograficznego (np. klucza).
- Rivest Shamir Adleman (RSA) – algorytm służący do szyfrowania i podpisywania danych. Najczęściej wykorzystywany z kluczami 1024 bitowymi i dłuższymi. Długość klucza silnie wpływa na czas obliczeń. Gdy używany jest RSA, podpisywanie jest wolniejsze niż sprawdzanie podpisu.
- Digital Signature Algorithm (DSA) – algorytm służący tylko do podpisywania (nie szyfrowania). Obsługuje maksymalną długość klucza 1024 bity. Gdy używany jest DSA, podpisywanie jest szybsze niż sprawdzanie podpisu.

Połączenie algorytmów symetrycznych i asymetrycznych

Ze względów wydajnościowych (kryptografia symetryczna jest dużo szybsza oraz oferuje większy poziom bezpieczeństwa przy tej samej długości klucza), często stosuje się połączenie obu technik. Kryptografia asymetryczna jest używana do uzgodnienia wspólnego klucza symetrycznego, który następnie jest używany do szyfrowania z użyciem algorytmów kryptografii symetrycznej.



1. Nadawca dysponuje certyfikatem (a tym samym kluczem publicznym adresata).
2. Nadawca generuje klucz symetryczny i używa go do zaszyfrowania danych.
3. Użyty klucz symetryczny jest szyfrowany kluczem publicznym odbiorcy (tylko odbiorca, z użyciem swojego klucza prywatnego, może go odszyfrować).
4. Zaszyfrowany klucz symetryczny i zaszyfrowane dane są przesyłane do odbiorcy.
5. Odbiorca używa swojego klucza prywatnego do odszyfrowania klucza symetrycznego.
6. Odbiorca używa otrzymanego klucza symetrycznego do odszyfrowania danych.

Struktura certyfikatu

Certyfikat zgodny ze standardem X.509 wersji 3 składa się, najczęściej, z:

- **Tematu (subject)** – Jednoznacznie identyfikującego właściciela (np. użytkownika, komputer, urządzenie sieciowe...) klucza prywatnego powiązanego z danym certyfikatem,
- **Dodatkowych informacji o właścicielu** (np. adres, itp.),
- Informacji o **podpisującym CA**,
- **Klucza publicznego**,
- Nazw **algorytmów kryptograficznych** używanych do stworzenia danego certyfikatu (np. do wygenerowania podpisu CA),
- Listy **rozszerzeń (extensions)** zawartych w certyfikacie,
- Informacji o sposobie uzyskiwania informacji o ewentualnym **odwołaniu certyfikatu** (np. adres www z którego można pobrać właściwą dla podpisującego CA listę CRL).

Rozszerzenia (extensions)

W wersji 3 standardu X.509 wprowadzono możliwość dodawanie dodatkowych informacji do certyfikatu, zwanych rozszerzeniami. Każde rozszerzenie składa się z 3 pól:

- **identyfikatora** jednoznacznie identyfikującego dane rozszerzenie,
- flagi określającej czy rozszerzenie jest **krytyczne** – jeśli tak, to aplikacja musi być w stanie zrozumieć znaczenie takiego rozszerzenia i wziąć je pod uwagę, aby być w stanie użyć danego certyfikatu. Rozszerzenia nie krytyczne mogą nie zostać zrozumiane przez aplikację, a mimo to certyfikat może zostać użyty.
- **wartości rozszerzenia** – znaczenie zależy od konkretnego rozszerzenia.

Do najważniejszych rozszerzeń należą:

- **Key usage** – określa ogólne zadania („niskopoziomowe”) w których certyfikat może być wykorzystany:
 - Digital signature – może być użyty od weryfikacji podpisu właściciela certyfikatu,
 - Non-repudiation – może być użyty do jednoznacznego określenia tożsamości podpisującego,
 - Key encipherment – może służyć do bezpiecznego przesyłania kluczy szyfrujących (np. symetrycznych), używanych następnie np. do (za/od)szyfrowania danych. Używane gdy wykorzystujemy mechanizmy RSA do zarządzania kluczami.
 - Data encipherment – może służyć do bezpośredniego szyfrowania danych z użyciem kryptografii asymetrycznej.
 - Key agreement – może służyć do uzgadniania klucza (np. symetrycznego), używanego następnie np. do (za/od)szyfrowania danych. Używane gdy wykorzystujemy mechanizmy DH (Diffie-Hellman) do zarządzania kluczami.
 - Key cert sign – może zostać użyty do sprawdzania poprawności podpisanego certyfikatu,
 - CRL Sign – może zostać użyty do weryfikacji podpisanej listy odwołań certyfikatów (CRL).
 - Encipher only – użyty w połączeniu z „Key agreement” oznacza, że uzyskany klucz symetryczny, może być wykorzystywany tylko do zaszyfrowywania danych.
 - Decipher only – użyty w połączeniu z „Key agreement” oznacza, że uzyskany klucz symetryczny, może być wykorzystywany tylko do odszyfrowywania danych.
 - **Extended key usage** – zawiera bardziej szczegółowe informacje o możliwości wykorzystania certyfikatu. Mają one postać listy identyfikatorów OID oznaczających różne zastosowania. Aplikacja może wymagać obecności określonych OID, aby wzięta pod uwagę dany certyfikat. Przykłady najpopularniejszych:
 - Client Authentication (uwierzytelnianie klienta) – 1.3.6.1.5.5.7.3.2
 - Server Authentication (uwierzytelnianie serwera) – 1.3.6.1.5.5.7.3.1
 - Secure e-Mail (zabezpieczenie poczty email) – 1.3.6.1.5.5.7.3.4
 - **CRL distribution point** – zawiera URL miejsca, skąd można pobrać listę CRL, w której należy szukać informacji o ewentualnym odwołaniu danego certyfikatu. System Windows pozwala na wykorzystanie protokołów HTTP, FTP i LDAP.
-

Proces tworzenia certyfikatu

1. Klient generuje klucze publiczny i prywatny.
2. Klucz prywatny jest najczęściej szyfrowany przed zapisaniem na dysku, karcie itp., co powoduje każdorazową konieczność podania hasła (najczęściej pełniącego rolę klucza szyfrującego w algorytmie symetrycznym) przed jego użyciem.

3. Klient tworzy żądanie certyfikatu (Certificate Signing Request - CSR),, łącąc:
 - a. klucz publiczny,
 - b. dodatkowe informacje identyfikacyjne, np. nazwę, kraj, miasto (najważniejszym parametrem jest **common name**)
 - c. pożądane rozszerzenia certyfikatu, np.: okres ważności, zbiór możliwych sposobów wykorzystania certyfikatu itp.
 4. Żądanie certyfikatu przesyłane jest do urzędu certyfikacji (CA), który:
 - a. sprawdza techniczną poprawność (format) żądania,
 - b. sprawdza czy polityka (policy) CA pozwala na podpisanie certyfikatu o podanych informacjach identyfikacyjnych (np. CA Politechniki Gdańskiej będzie najprawdopodobniej podpisywało tylko certyfikaty, w których organizacja właściciela jest określona jako „Politechnika Gdańska”) – żądanie zostaje zaakceptowane lub odrzucone,
 - c. sprawdza czy żądane przez klienta rozszerzenia certyfikatu są zgodne z dopuszczalnymi:
 - i. część rozszerzeń z żądania może zostać umieszczona w certyfikacie bez zmian,
 - ii. część może zostać usunięta/zmodyfikowana, a nowe rozszerzenia dopisane do generowanego certyfikatu.
 - d. klucz publiczny wraz z dodatkowymi informacjami i rozszerzeniami zostaje podpisany przez CA jej kluczem prywatnym, tworząc certyfikat.
 5. Klient otrzymuje certyfikat, który jest potem wykorzystywany w połączeniu z jego kluczem prywatnym.
-

Sprawdzanie poprawności certyfikatu

Certyfikat może zostać użyty, jeśli jesteśmy w stanie wywnioskować, że rzeczywiście należy oczekiwanego właściciela i zawiera prawidłowe informacje i klucz publiczny. Aby móc to stwierdzić opieramy się na tzw. łańcuchu certyfikatów. Aby go stworzyć, odczytujemy z badanego certyfikatu informację o CA które go podpisało i pobieramy z kolei certyfikat tego CA. Jeśli dane CA nie jest root-CA (tzn. urzędem certyfikacji, którego tożsamości nikt inny już nie poświadcza), to pobieramy z kolei certyfikat CA, które wystawiło certyfikat aktualnie badanemu CA. Powtarzamy tę czynność, aż dojdziemy do certyfikatu CA, którego nie poświadcza już nikt inny – jest to tzw. root-CA, a certyfikat taki jest typu self-signed (to CA nie prosiło nikogo o poświadczenie swojego certyfikatu, tylko samo go sobie podpisało). Stworzyliśmy w ten sposób łańcuch wzajemnych zależności, z których wynika, że jeśli root-CA jest godne zaufania, to badany certyfikat też jest. W związku z tym, aplikacja lub system operacyjny powinien oferować możliwość określenia którym root-CA należy zaufać, a którym nie – np. w systemie Windows XP, ufa się root-CA, których certyfikatu umieszczono w magazynie „**Zaufane główne urzędy certyfikacji**” (patrz rozdział: „**Zarządzanie certyfikatami (Windows XP)**”).

Szczegółowa procedura weryfikacji certyfikatu:

1. Otrzymany certyfikat sprawdzany jest pod względem podstawowej poprawności formatu.
2. Tworzony jest łańcuch certyfikatów (certificate chain), zawierający sprawdzany certyfikat, certyfikat CA które wystawiło dany certyfikat, oraz (jeśli jest to CA pośrednie) to również certyfikaty wszystkich CA nadrzędnych, aż do root CA (typu self-signed).
3. Dla całego łańcucha:

- a. Sprawdzane są krytyczne rozszerzenia certyfikatu (własności które muszą być rozumiane przez aplikację) – jeśli występuje jakaś niezrozumiała, certyfikat jest odrzucany.
 - b. Sprawdzane są rozszerzenia certyfikatu, aby sprawdzić, czy jest on możliwy do użycia w roli w której próbuje go wykorzystać aplikacja (np. do podpisywania),
 - c. Sprawdzane są listy CRL, w celu weryfikacji, czy żaden z certyfikatów nie został odwołany.
 - d. Sprawdzana jest poprawność podpisów CA, w celu wykrycia ewentualnych, nieuprawnionych modyfikacji certyfikatów.
 - e. Sprawdzany jest okres ważności certyfikatu.
- 4. Następuje sprawdzenie czy łańcuch certyfikatów kończy się znanym, zaufanym certyfikatem root-CA (typu self-signed). Oznacza to, że jeśli ufamy root-CA, a łańcuch jest poprawny i nie przerwany aż do badanego certyfikatu, to jemu też możemy zaufać.**
-

Listy odwołań certyfikatów (Certificate Revocation List – CRL)

Jest to, podpisana przez CA, lista numerów seryjnych certyfikatów, które z różnych powodów zostały **odwołane (revoked)** przed końcem ich przypisanego okresu ważności. Lista CRL jest generowana przez CA wyłącznie dla certyfikatów podpisanych przez dane CA.

Każdy z odwołanych numerów seryjnych opatrzony jest **powodem odwołania**, np.:

- Key compromise – klucz prywatny powiązany z danym certyfikatem został ujawniony,
- CA compromise – klucz prywatny podpisującego CA został ujawniony,
- Affiliation changed – właściciel certyfikatu przestał być częścią wystawiającej organizacji,
- Superseded – certyfikat został zastąpiony uaktualnionym,
- Cessation of operation – właściciel certyfikatu (np. serwer WWW) zaprzestał działalności,
- Certificate hold – powoduje tymczasowe ZAWIESZENIE ważności certyfikatu. Jedyny przypadek, gdy certyfikat może odzyskać ważność.

Występują **2 rodzaje list CRL**:

- bazowe (base CRL) – lista obejmuje wszystkie certyfikaty odwołane przez dane CA, które zostały podpisane jego obecnie obowiązującym kluczem prywatnym.
- różnicowe (delta CRL) – lista obejmuje wyłącznie certyfikaty odwołane od czasu wygenerowania ostatniej listy bazowej.

Lista CRL powinna być sprawdzana przez aplikację (lub okresowo przez system operacyjny i wyniki udostępniane aplikacjom) zanim certyfikat zostanie uznany za ważny – jednak w rzeczywistości często krok ten nie jest realizowany.

S/MIME (Secure / Multipurpose Internet Mail Extension)

S/MIME jest standardem opisującym sposób zabezpieczania wiadomości w formacie MIME za pomocą mechanizmów kryptografii asymetrycznej. Pozwala na realizację:

- kontroli integralności – zabezpieczenie przed modyfikacją,
- zachowania niezaprzeczalności – możliwość jednoznacznego określenia nadawcy,
- zachowania poufności – szyfrowanie zawartości.

Format MIME pozwala na przesyłanie wiadomości e-mail złożonych z wielu elementów (tekst, obraz, dźwięk itp.), w tym elementów w binarnych – opatrując je odpowiednimi nagłówkami

umożliwiający ich późniejszą interpretację po stronie odbiorczej i konwertując na odpowiedni format (np. tekstowy).
Standard ten jest też używany w wielu innych przypadkach, gdy taka funkcjonalność jest konieczna.

Polecenia dodatkowe

time <polecenie>

Powoduje wykonanie polecenia podanego jako parametr i podanie czasu jego wykonywania.

Podawane są 3 wartości:

- **real** – rzeczywisty, całkowity czas wykonania polecenia,
 - **user** – czas zajęcia procesora przez wykonywany kod polecenia,
 - **sys** – czas zajęcia procesora przez funkcje systemu operacyjnego, konieczne do wykonania polecenia.
-

OpenSSL

OpenSSL to zestaw narzędzi kryptograficznych implementujący protokoły sieciowe Secure Sockets Layer (SSL v2/v3) i Transport Layer Security (TLS v1) oraz wymagane przez nie standardy kryptograficzne.

Program openssl z kolei, to narzędzie wiersza poleceń przeznaczone do używania różnych funkcji kryptograficznych biblioteki OpenSSL. Można go używać do:

- Tworzenia kluczy RSA, DH i DSA.
- Wystawiania certyfikatów X.509, CSR oraz CRL.
- Obliczania skrótów wiadomości.
- Szyfrowania i deszyfrowania.
- Testowania klientów i serwerów SSL/TLS.
- Obsługi poczty z podpisem S/MIME lub zaszyfrowanej.

Obsługiwane formaty plików

OpenSSL obsługuje dwa podstawowe formaty plików:

- **PEM** (Privacy-enhanced Mail) – domyślny, tekstowy, może zawierać wiele certyfikatów w jednym pliku, rozszerzenia: .pem, .crt
- **DER** – opcjonalny, binarny, jeden plik zawiera jeden certyfikat, rozszerzenia: .der, .cer

Domyślnie stosowany jest format PEM. Jeśli chcemy użyć/wygenerować plik w formacie DER, musimy użyć odpowiedniej opcji linii polecenia, np.:

- do wyznaczenia formatu zapisu: **-outform DER**
- do określenia formatu odczytu: **-inform DER**

Składnia

```
openssl <polecenie> [ <opcje_polecenia> ]
```

Polecenia:

- **help** – wypisuje listę poleceń,

- **req** – służy do generowania kluczy prywatnych i żądań certyfikatów (Certificate Signing Request - CSR) w formacie X.509,
- **ca** – służy do obsługi urzędu certyfikacji, wymaga poprawnie skonfigurowanego pliku `openssl.cnf`
- **pkcs12** – pozwala na obsługę formatu PKCS#12, używanego do importu i eksportu certyfikatów wraz z kluczami prywatnymi,
- **x509** – umożliwia zarządzanie certyfikatami X.509,
- **enc** – pozwala na szyfrowanie i rozszyfrowywanie,
- **rsautl** – pozwala na bezpośrednie wykorzystanie kryptografii asymetrycznej do szyfrowania i podpisywania niewielkich ilości danych,
- **smime** – funkcje pozwalające na obsługę poczty w formacie S/MIME,

Polecenie req

Polecenie **req** służy do generowania żądania certyfikatu (na które składa się klucz publiczny wraz z dodatkowymi informacjami) oraz klucza prywatnego. Domyślnie wszystkie pliki odczytywane i generowane przez polecenie są w formacie PEM.

Polecenie korzysta z **pliku konfiguracyjnego** OpenSSL (domyślnie ***openssl.cnf***), lecz nie jest on konieczny, gdyż wszystkie niezbędne opcje można podać z linii polecenia.

```
openssl req [opcje]
```

Opcje:

Opcje można podzielić na kilka grup, odpowiedzialnych za różne elementy żądania.

Źródło żądania certyfikatu (WYMAGANE jedno z):

-new – oznacza tworzenie nowego żądania certyfikatu,

-x509 – oznacza utworzenie gotowego, podpisanego własnym kluczem prywatnym (self-signed) certyfikatu, zamiast żądania certyfikatu.

-in <plik> – oznacza pobranie już istniejącego żądania z podanego pliku.

Miejsce docelowe zapisu żądania certyfikatu:

-out <plik> – opcjonalna – pozwala na podanie pliku do którego trafi wygenerowane żądanie certyfikatu (przy opcji **-new**) lub podpisany własnym kluczem prywatnym certyfikat (przy opcji **-x509**).

-noout – opcjonalna – nigdzie nie zapisuj ani nie wyświetlaj żądania certyfikatu. Przydatne gdy przeprowadzamy analizę już istniejącego żądania, wczytując je opcją **-in**.

Jeśli nie podamy opcji **-out** ani **-noout**, dane trafią na ***stdout*** (konsolę).

Klucz prywatny:

-newkey rsa:<bity> – opcjonalna – oznacza wygenerowanie nowego klucza tajnego RSA o podanej w bitach długości (alternatywą jest użycie istniejącego – patrz opcja **key**). Jeśli nie podamy liczby bitów, zostanie przyjęta wartość domyślna z pliku konfiguracyjnego.

-key <plik> – opcjonalna – powoduje użycie JUŻ ISTNIEJĄCEGO klucza prywatnego zawartego w pliku o podanej nazwie.

Jeśli nie podamy ani **-newkey** ani **-key** to w przypadku **-x509** program będzie czekał na klucz prywatny podany przez ***stdin*** (konsolę), a w przypadku **-new** zostanie wygenerowany klucz prywatny RSA o liczbie bitów podanej w pliku konfiguracyjnym.

-keyout <plik> – opcjonalna – pozwala na podanie pliku do którego trafi wygenerowany klucz prywatny. Jeśli nie podamy tej opcji, klucz prywatny zostanie zachowany z nazwą określoną w pliku konfiguracyjnym.

Opcje dodatkowe:

-nodes – opcjonalna – pozwala uniknąć zapisywania klucza prywatnego w postaci zaszyfrowanej. Domyślnie klucz prywatny zapisywany jest w postaci zaszyfrowanej i do jego każdorazowego użycia niezbędne jest podawanie hasła (klucza szyfrującego) – ta opcja pozwala tego uniknąć kosztem oczywistego zmniejszenia bezpieczeństwa.

-config <plik> – opcjonalna – pozwala na podanie z jakiego pliku konfiguracyjnego korzystać, zamiast domyślnego openssl.cnf

Opcje analizy (stosowane przy pobraniu żądania przy użyciu opcji -in):

Jeśli nie użyjemy opcji **-noout**, do wyniku polecenia zostanie dołączona treść żądania.

-text – podaje szczegóły żądania (informacje dot. tożsamości, żądanych właściwości, dodatkowych parametrów, możliwości użycia itp.).

-pubkey – wyświetla sam klucz publiczny.

-verify – sprawdza techniczną poprawność żądania.

Przykłady:

```
openssl req -new -newkey rsa:512 -keyout klucz_pr.pem -out zadanie_cert.req
```

Wygenerowanie nowego żądania certyfikatu (zapisanego w pliku zadanie_cert.req) i nowego klucza prywatnego RSA o długości 512 bitów (zapisanego w zaszyfrowanym pliku klucz_pr.pem).

```
openssl req -new -key klucz_pr.pem -out zadanie_cert.req
```

Wygenerowanie nowego żądania certyfikatu (zapisanego w pliku zadanie_cert.req) na podstawie już istniejącego klucza prywatnego wczytanego z pliku klucz_pr.pem

```
openssl req -x509 -key klucz_pr.pem -out zadanie_cert.req
```

Wygenerowanie nowego, podpisanego przez samego siebie certyfikatu (zapisanego w pliku zadanie_cert.req) na podstawie już istniejącego klucza prywatnego wczytanego z pliku klucz_pr.pem

```
openssl req -in zadanie_cert.req -noout -text
```

Wyświetla szczegółowe informacje o żądaniu certyfikatu zawartym w pliku zadanie_cert.req.

Polecenie ca

Polecenie służy do obsługi **urzędu certyfikacji (Certificate Authority – CA)**, pozwalającego na, między innymi, podpisywanie żądań certyfikatów (czyli tworzenie certyfikatów na podstawie żądań) oraz odwoływanie certyfikatów i tworzenie list CRL (Certificate Revocation List).

Polecenie WYMAGA poprawnie skonfigurowanego **pliku konfiguracyjnego**, gdyż wszystkich koniecznych parametrów nie da się podać z linii poleceń.

```
openssl ca [opcje]
```

W pliku konfiguracyjnym można zdefiniować wiele niezależnych CA. Zawarty na jego początku parametr **default_ca** określa domyślnie używane CA. Jeśli chcemy, wydając dane polecenie, użyć CA innego niż domyślne, robimy to używając opcji: **-name <nazwa_CA>**

Opcje dla podpisywania certyfikatu:

Opcje wczytania żądania certyfikatu:

-in <plik> – pozwala na wczytanie żądania certyfikatu z podanego pliku. Jeśli nie podamy tej opcji, polecenie będzie czekało na podanie żądanie certyfikatu ze **stdin** (konsoli).

Opcje polityki:

-policy <nazwa_polityki> – nakazuje sprawdzenie, czy żądanie certyfikatu spełnia warunki polityki o podanej nazwie. Jeśli nie podamy tej opcji, to zastosowana zostanie polityka domyślna, określona w pliku konfiguracyjnym.

Opcje właściwości certyfikatu:

-days <dni> – ustala ważność certyfikatu na określoną liczbę dni, poczynając od chwili obecnej.

-startdate <YYMMDDHHMMSS> – ustala początek ważności certyfikatu.

-enddate <YYMMDDHHMMSS> – ustala koniec ważności certyfikatu.

-extensions <sekcja> – powoduje użycie dla certyfikatu właściwości określonych w sekcji o podanej nazwie zamiast w sekcji domyślnej (określonej w pliku konfiguracyjnym parametrem **x509_extensions**).

-extfile <plik> – pozwala na użycie w opcji **-extensions**, sekcji znajdującej się w pliku o podanej nazwie, zamiast w pliku konfiguracyjnym.

Miejsce docelowe zapisu certyfikatu:

-out <plik> – nakazuje zapisać certyfikat do podanego pliku. Jeśli nie podamy tej opcji, certyfikat zostanie wysłany na **stdout** (konsolę).

Opcje dla odwoływania certyfikatów i generacji list CRL:

-revoke <plik> – nakazuje odwołanie certyfikatu zawartego w podanym pliku. Można się tu posłużyć plikami zgromadzonymi w odpowiednim katalogu CA.

-genrl – powoduje wygenerowanie listy CRL o kolejnym numerze. Domyślnie lista wysyłana jest na **stdout** (konsolę), można jednak użyć opcji **-out**, aby zapisać ją do pliku.

-updatedb – powoduje przejście bazy danych wystawionych przez CA certyfikatów i zaznaczenie tych, których czas ważności upłynął.

Opcje informacyjne:

-status <nr_seryjny> – wyświetla aktualny stan certyfikatu o podanym numerze seryjnym.

Przykłady:

```
openssl ca -in cert.req -out cert.cer
```

Tworzy certyfikat na podstawie żądania zawartego w pliku cert.req, jeśli spełnia ono domyślną politykę, a następnie zapisuje certyfikat w pliku cert.cer.

```
openssl ca -in cert.req -days 365 -policy moja_polityka
```

Tworzy certyfikat, ważny 365 dni, na podstawie żądania zawartego w pliku cert.req, jeśli spełnia ono politykę moja_polityka (zdefiniowaną w pliku konfiguracyjnym), a następnie wyświetla certyfikat na konsoli.

```
openssl ca -in cert.req -out cert.cer -extensions usr_cert
```

Tworzy certyfikat, o właściwościach opisanych w sekcji usr_cert (w pliku konfiguracyjnym), na podstawie żądania zawartego w pliku cert.req, jeśli spełnia ono domyślną politykę, a następnie zapisuje certyfikat w pliku cert.cer.

```
openssl ca -in cert.req -out cert.cer -extfile definicje.txt -extensions srv_cert
```

Tworzy certyfikat, o właściwościach opisanych w sekcji `srv_cert` (w pliku `definicje.txt`), na podstawie żądania zawartego w pliku `cert.req`, jeśli spełnia ono domyślną politykę, a następnie zapisuje certyfikat w pliku `cert.cer`.

```
openssl ca -revoke 1.pem
```

Unieważnia certyfikat określony plikiem `1.pem`.

```
openssl ca -gencrl -out lista.crl
```

Generuje nową listę CRL i zapisuje ją do pliku `lista.crl`.

Polecenie x509

Polecenie `x509` służy do **manipulacji certyfikatami** w formacie X.509. Posiada wiele funkcji, poczynając od wyświetlania informacji o certyfikatach, poprzez zmianę formatów ich zapisu, a kończąc na ich podpisywaniu (coś w rodzaju mini-CA).

```
openssl x509 [opcje]
```

Zastosowanie informacyjne

W zastosowaniu informacyjnym odczytujemy certyfikat ze **stdin** (konsoli - domyślnie) lub z pliku z użyciem opcji **-in**.

Dalej podajemy jedną z opcji powodujących wyświetlenie interesujących nas informacji.

Domyślnie polecenie dołącza do wyniku treść wczytanego certyfikatu – jeśli nam to nie odpowiada, używamy opcji **-noout**.

Opcje wczytywania i zapisu danych:

-in <plik> – pozwala na wczytanie certyfikatu z pliku o podanej nazwie. Jeśli nie podamy tej opcji, polecenie będzie czekać na wprowadzenie certyfikatu z `stdin` (konsoli).

-noout – powoduje rezygnację z wyświetlenia/zapisania treści certyfikatu. Przydatne gdy wyświetlamy informację dla certyfikatu wczytanego opcją **-in**, gdyż domyślnie zostałby on również dołączony do wyniku polecenia.

Opcje informacyjne:

-text – podaje szczegółowe informacje o certyfikacie (informacje dot. tożsamości, właściwości, dodatkowych parametrów, możliwości użycia itp.).

-dates – podaje daty ważności certyfikatu.

-purpose – podaje możliwe zastosowania certyfikatu.

-pubkey – wyświetla sam klucz publiczny zawarty w certyfikacie.

-serial – wyświetla numer seryjny certyfikatu.

-subject – wyświetla temat certyfikatu (czyli jednoznaczny identyfikację jego właściciela).

-issuer – wyświetla dane wystawcy certyfikatu (CA).

Zastosowanie w podpisywaniu certyfikatów

Możliwe są 2 metody: z użyciem opcji **-signkey**, lub zbioru 3 opcji: **-CA**, **-CAkey**, **-CAserial**.

W przypadku każdej z nich:

- danymi wejściowymi może być żądanie certyfikatu (należy użyć opcji **-req**) lub podpisany już certyfikat,
- ostateczną postać certyfikatu można modyfikować opcjami właściwości certyfikatu podanymi poniżej (**-serial**, **-crltext**, **-days**, **-extensions**, ...).

Opcje wczytywania i zapisu danych:

-req – opcja oznacza, że wprowadzone dane to żądanie certyfikatu do podpisania. W przeciwnym razie oczekiwany jest certyfikat.

-in <plik> – pozwala na wczytanie żądania/certyfikatu z pliku o podanej nazwie. Jeśli nie podamy tej opcji, polecenie będzie czekać na wprowadzenie tych danych z **stdin** (konsoli).

-out <plik> – pozwala na zapisanie certyfikatu do pliku. Jeśli nie podamy tej opcji, certyfikat zostanie wysłany na **stdout** (konsolę).

Opcje podpisywania:

Pierwsza metoda:

-signkey <plik> – w zależności czy dane wprowadzone ze **stdin** lub pliku, to:

- **żądanie certyfikatu** – spowoduje podpisanie żądania kluczem prywatnym zawartym w podanym pliku. Informacja o CA, które podpisało certyfikat, zostanie ustawiona taką samą wartością jak informacja o tożsamości zawarta w certyfikacie. Będzie to więc certyfikat typu self-signed. Właściwości z żądania kopiowane są do certyfikatu bez zmian.
- **podpisany już certyfikat** – spowoduje zmianę informacji o podpisującym jak powyżej, zmianę czasu ważności na 1 miesiąc od chwili obecnej i podpisanie certyfikatu podanym kluczem prywatnym. Właściwości i inne informacje przeniesione będą bez zmian. Będzie to więc zmiana certyfikatu na self-signed.

Druga metoda – pozwala na podpisanie żądania z użyciem istniejącego CA lub zmianę CA które podpisało certyfikat. Numer seryjny zawarty w pliku zostanie zwiększony, lecz certyfikat nie zostanie dopisany do bazy danych wystawionych certyfikatów (pliki index.txt i katalog newcerts).

-CA <plik> – pozwala na podanie pliku zawierającego certyfikat CA.

-CAkey <plik> – pozwala na podanie pliku zawierającego klucz prywatny CA.

-CAserial <plik> – pozwala na podanie pliku zawierającego numer seryjny dla certyfikatu.

Opcje właściwości certyfikatu:

-serial <numer> – ręcznie ustawia numer seryjny certyfikatu na podany numer.

-clrext – usuń ustawione w żądaniu certyfikatu właściwości. Domyślnie są one kopiowane do podpisanego certyfikatu.

-days, -extensions, -extfile – ustawia właściwości certyfikatu, jak w przypadku polecenia **ca**.

Przykłady:

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

Wyświetla pełne informacje o certyfikacie z pliku cert.pem, bez treści samego certyfikatu (-noout).

```
openssl x509 -in cert.pem -purpose
```

Wyświetla możliwości użycia certyfikatu z pliku cert.pem, oraz samą jego treść.

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

Wyświetla unikalny identyfikator właściciela (subject) certyfikatu z pliku cert.pem, bez treści samego certyfikatu (-noout).

```
openssl x509 -req -in careq.pem -extfile openssl.cnf -extensions v3_ca  
-signkey key.pem -out cacert.pem
```

Tworzy certyfikat z żądania zawartego w pliku careq.pem, dołączając rozszerzenia zawarte w sekcji v3_ca pliku openssl.cnf (domyślnie opisuje ona certyfikat możliwy do wykorzystania jako certyfikat CA). Certyfikat jest podpisywany kluczem z pliku key.pem i zapisywany do pliku cacert.pem.

```
openssl x509 -req -in req.pem -extfile openssl.cnf -extensions v3_usr -CA  
cacert.pem -CAkey key.pem -CA serial
```

Tworzy certyfikat z żądania zawartego w pliku `careq.pem`, dołączając rozszerzenia zawarte w sekcji `v3_usr` pliku `openssl.cnf` (domyślnie opisuje ona certyfikat kliencki). Certyfikat jest podpisywany przez CA (patrz „Struktura folderów/plików CA opartego na pakiecie OpenSSL”), które przechowuje swój certyfikat w pliku `ca.cert.pem`, klucz prywatny w pliku `key.pem`, a aktualny numer seryjny wystawianego certyfikatu w pliku `serial`.

Polecenie `enc`

Polecenie służy do **zaszyfrowania** (jeśli podano opcję `-e`) lub **rozszyfrowania** (jeśli podano opcję `-d`) danych z **użyciem kryptografii symetrycznej**. Jeśli nie podamy ani `-e` ani `-d`, to domyślnie przyjęte zostanie zaszyfrowanie danych.

Domyślnie dane wprowadzane są ze **`stdin`** i po przetworzeniu wysyłane do **`stdout`**. Zachowanie to można zmienić podając opcje `-in <plik>` (określa plik wejściowy) i/lub `-out <plik>` (określa plik wyjściowy).

Jako, że w wyniku szyfrowania uzyskujemy plik binarny, zawierający kody niemożliwe do wyświetlenia na konsoli w formie zrozumiałych znaków, a także trudne do przesłania niektórymi metodami (np. z użyciem starszych serwerów pocztowych), możemy zastosować kodowanie **Base64** (mające na celu odwracalne przekształcenie takich danych do formatu tekstowego). W tym celu stosujemy opcję `-base64`. Jeśli ją podamy, to:

- w przypadku szyfrowania, wynik zostanie poddany kodowaniu base64,
- w przypadku odszyfrowywania, dane wejściowe zostaną poddane dekodowaniu base64.

Należy też podać wybrany **algorytm szyfrowania** - jeśli tego nie zrobimy, nie zostanie wykonane żadne przekształcenie wprowadzonych danych (na wyjściu pojawią się te same dane, które wprowadziliśmy na wejściu). Listę możliwych algorytmów otrzymamy w wyniku wydania polecenia:

`openssl enc help`

`openssl enc [opcje]`

Opcje:

`-in <plik>`— wczytaj dane do przetworzenia z pliku `<plik>`. Jeśli nie podamy tej opcji, dane zostaną odczytane z **`stdin`** czyli konsoli.

`-out <plik>`— zapisz przetworzone dane w pliku `<plik>`. Jeśli nie podamy tej opcji, dane zostaną zapisane na **`stdout`** czyli na konsoli.

`-pass <klucz>` – umożliwia podanie klucza szyfrującego w linii polecenia (patrz „**Sposoby podawania kluczy w linii poleceń**”). Jeśli tego nie zrobimy, to program zażąda od nas wprowadzenia klucza w sposób interaktywny.

`-e` – oznacza zaszyfrowanie danych wejściowych. Jeśli nie podamy opcji `-e` ani `-d`, to domyślnie zostanie przyjęte, że chcemy zaszyfrować dane.

`-d` – oznacza odszyfrowanie danych wejściowych. Jeśli jej nie podamy, domyślnie przyjmowane jest, że chcemy zaszyfrować dane.

`-base64` – dane zostaną poddane przekształceniu base64 po zaszyfrowaniu lub przed rozszyfrowaniem.

`-salt` – powoduje użycie dodatkowych, losowych danych, przy tworzeniu klucza szyfrującego z podanego przez użytkownika hasła. Dane te są następnie przesyłane wraz z wiadomością. Bez tej funkcji, takie same hasła, dawały by zawsze w efekcie takie same klucze szyfrujące, co znacznie obniża bezpieczeństwo. Opcja `-salt` powinna być używana zawsze, chyba że używamy wersji OpenSSL która nie obsługuje tej funkcji.

Przykłady:

`openssl enc -des3 -salt -in file.txt -out file.des3`

Zaszyfrowanie pliku file.txt mechanizmem 3DES i zapisanie wyniku do pliku file.3des. Użyty mechanizm salt.

```
openssl enc -des3 -salt -in file.txt -out file.des3 -base64
```

Jak wyżej, lecz plik wynikowy będzie w formacie tekstowym (a nie binarnym) dzięki przekształceniu base64.

```
openssl enc -des3 -d -salt -in file.des3 -out file.txt -k mypassword
```

Odszyfrowanie pliku file.3des mechanizmem 3DES i zapisanie wyniku do pliku file.txt. Użyty mechanizm salt i hasło mypassword.

Polecenie pkcs12

Polecenie służy do **obsługi formatu PKCS#12**, który pozwala na eksport i import paczek, zawierających **certyfikaty i klucze prywatne**. Użycie tego formatu jest jednym z podstawowych sposobów przenoszenia certyfikatów klienckich wraz z ich kluczami prywatnymi i instalowania ich u klientów.

Opcje:

-export – oznacza eksport certyfikatów/kluczy. Domyślnie przyjmuje się import.

-inkey <plik> – pozwala na określenie pliku zawierającego klucz prywatny do eksportu

-in <plik> – przy eksporcie (gdy użyjemy opcji **-export**) pozwala na podanie pliku zawierającego certyfikat do eksportu; przy imporcie (bez opcji **-eksport**) pozwala na określenie nazwy pliku PKCS#12 do importu/weryfikacji.

-certfile <plik> - przy eksporcie (gdy użyjemy opcji **-eksport**), pozwala na dołączenie do eksportowanych danych certyfikatów z podanego pliku.

-out <plik> – przy eksporcie (gdy użyjemy opcji **-eksport**) pozwala na określenie nazwy pliku PKCS#12 do którego eksportujemy; przy imporcie (bez opcji **-eksport**) pozwala na określenie pliku do którego zapisany zostanie klucz prywatny i certyfikaty.

-name „<nazwa>” – przy eksporcie (gdy użyjemy opcji **-eksport**) pozwala na nadanie zestawowi certyfikat/klucz_prywatny przyjaznej dla użytkownika nazwy, pod którą będzie on widoczny dla użytkownika w systemie operacyjnym.

-info – podaje informacje o pliku PKCS#12 określonym opcją **-in**. Jeśli nie podamy opcji **-noout**, to wynik będzie zawierał certyfikaty oraz klucze prywatne w formie znaków ASCII. Przed wyświetleniem klucza prywatnego, program zapyta o klucz szyfrujący (hasło) i wyświetli klucz prywatny w formie zaszyfrowanej – aby wyświetlić go w formie niezasyfrowanej należy użyć opcji **-nodes**.

-nodes – pozwala uniknąć szyfrowania klucza prywatnego przy importowaniu go z pliku PKCS#12 i zapisywaniu oraz przy wyświetlaniu opcją **-info**.

-noout – powoduje, że program nie zapisuje ani nie wyświetla treści certyfikatów i kluczy prywatnych. Przydatne np. przy testach, weryfikacji poprawności danych itp.

Polecenie rsautl

Polecenie rsautl pozwala bezpośrednio wykorzystać mechanizmy kryptografii asymetrycznej do szyfrowania i podpisywania niewielkich partii danych. Jak łatwo się domyślić, konkretny zestaw wykorzystywanych mechanizmów, to mechanizmy RSA.

Polecenie nie obsługuje dużych porcji danych – do ich obsługi należy użyć polecenia **smime**, które łączy kryptografię asymetryczną (stosowaną do uzgodnienia klucza symetrycznego) i symetryczną (stosowaną do samego szyfrowania).

Opcje wejścia-wyjścia:

-in <plik> – opcjonalna – wczytaj dane do przetworzenia z pliku **<plik>**. Jeśli nie podamy tej opcji, dane zostaną odczytane z **stdin** czyli konsoli.

-out <plik> – opcjonalna – zapisz przetworzone dane w pliku **<plik>**. Jeśli nie podamy tej opcji, dane zostaną zapisane na **stdout** czyli na konsoli.

Opcje pobierania klucza:

-inkey <plik> – nakazuje użycie **klucza prywatnego** pobranego z podanego pliku.

-pubin <plik> – nakazuje użycie **klucza publicznego** pobranego z podanego pliku.

-certin <plik> – nakazuje użycie **klucza publicznego pobranego z certyfikatu** wczytanego z podanego pliku.

Opcje wyboru działania:

-sign – podpisz dane.

-verify – zweryfikuj podpisane dane.

-encrypt – zaszyfruj dane.

-decrypt – odszyfruj dane.

Przykłady:

```
openssl rsautl -sign -in file -inkey key.pem -out sig
```

Podpisanie pliku file przy użyciu klucza prywatnego zawartego w pliku key.pem i zapisanie wyniku do pliku sig.

Polecenie smime

Polecenie służy do obsługi wiadomości w formacie S/MIME – patrz opis w „S/MIME (Secure / Multipurpose Internet Mail Extension)”. Format ten funkcjonuje w oparciu o opisane wcześniej połączenie kryptografii symetrycznej i asymetrycznej.

Należy pamiętać, że:

- Do **podpisania** potrzeba:
 - certyfikatu podpisującego – gdyż jest źródłem danych o osobie która podpis wykonała, które zostaną umieszczone w dokumencie
 - jego klucza prywatnego do samego wykonania podpisu.
- Do **weryfikacji podpisu** potrzeba:
 - certyfikatu podpisującego – bo zawiera klucz publiczny konieczny do sprawdzenia podpisu,
 - certyfikatu CA, które wystawiło certyfikat podpisującego – do jego weryfikacji i sprawdzenia czy jest autentyczny.
- Do **zaszyfrowania** danych potrzeba:
 - klucza publicznego odbiorcy (a więc jego certyfikatu),
- Do **odszyfrowania**:

- o klucza prywatnego odbiorcy.

Opcje wyboru działania:

- sign** – podpisz dane.
- verify** – zweryfikuj podpisane dane.
- encrypt** – zaszyfruje dane.
- decrypt** – odszyfruj dane.

Opcje wejścia-wyjścia:

- in <plik>** – opcjonalna – wczytaj dane do przetworzenia z pliku **<plik>**. Jeśli nie podamy tej opcji, dane zostaną odczytane z **stdin** czyli konsoli.
- out <plik>** – opcjonalna – zapisz przetworzone dane w pliku **<plik>**. Jeśli nie podamy tej opcji, dane zostaną zapisane na **stdout** czyli na konsoli.

Opcje pobierania klucza stosowane przy podpisywaniu:

- inkey <plik>** – przy nakazuje użycie **klucza prywatnego** pobranego z podanego pliku.
- signer <plik>** – nakazuje użycie **klucza publicznego pobranego z certyfikatu** wczytanego z podanego pliku.

Opcje pobierania klucza stosowane przy weryfikacji podpisu:

- signer <plik>** – nakazuje użycie **klucza publicznego pobranego z certyfikatu** wczytanego z podanego pliku.
- CAfile <plik>** - pozwala na podanie **certyfikatu CA**, które wystawiło certyfikat podpisującego wiadomość.

Opcje wyboru algorytmu szyfrowania symetrycznego (używane przy zaszyfrowywaniu - **encrypt**):

- des <plik>**, -**des3 <plik>**, -**rc2-40 <plik>**, -**rc2-64 <plik>**, -**rc2-128 <plik>**, -**aes128 <plik>**, -**aes196 <plik>**, -**aes256 <plik>**

Po dowolnej z tych opcji należy podać plik zawierający certyfikat adresata. Klucz symetryczny (konieczny do szyfrowania wybraną metodą) zostanie automatycznie wygenerowany i zabezpieczony z użyciem klucza publicznego adresata.

Opcje pobierania klucza (stosowane odszyfrowywaniu -**decrypt**):

- inkey <plik>** – stosowane przy nakazuje użycie **klucza prywatnego** pobranego z podanego pliku.

Przykłady:

```
openssl smime -sign -in plik1.txt -signer cert.pem -inkey cert.key
```

Podpisuje dane pobrane z pliku **plik1.txt** z użyciem certyfikatu i klucza prywatnego nadawcy. Certyfikaty pobierany jest z pliku **cert.pem**, a klucz prywatny z pliku **cert.key**. Podpis ma formę dopisku do oryginalnej wiadomości.
Wynik trafia na konsolę.

```
openssl smime -sign -in plik1.txt -signer cert.pem -inkey cert.key -nodetach
```

Jak wyżej, lecz cała wiadomość i podpis mają scaloną, nieczytelną bez weryfikacji podpisu, formę.

```
openssl smime -verify -in plik.sig -signer cert.pem -CAfile CA.pem
```

Powoduje weryfikację podpisanego pliku **plik.sig**, z użyciem certyfikatu osoby podpisującej zawartego w pliku **cert.pem**, oraz certyfikatu CA (pobieranego z pliku **CA.pem**) które wystawiło mu ten certyfikat.

Sprawdzana jest jego integralność (brak modyfikacji), fakt, czy rzeczywiście podpisał to właściciel danego certyfikatu, oraz czy dany certyfikat został rzeczywiście wystawiony przez określone CA.

```
openssl smime -encrypt -in plik1.txt -des3 cert.pem
```

Szyfruje zawartość pliku **plik1.txt** z użyciem algorytmu **3des**. Konieczny klucz symetryczny jest generowany automatycznie, dołączany do wiadomości i szyfrowany z użyciem klucza publicznego zawartego w certyfikacie odczytywanym z pliku **cert.pem**. Wynik trafia na konsolę.

```
openssl smime -decrypt -in plik1.txt -inkey priv.key
```

Odszyfrowuje dane zawarte w pliku **plik1.txt**. Informacje o użytym algorytmie szyfrowania i kluczu szyfrującym rozszyfrowywane są najpierw, z użyciem kryptografii asymetrycznej i klucza tajnego z pliku **priv.key**. Następnie, z użyciem tych informacji i algorytmu symetrycznego, odszyfrowywana jest sama treść wiadomości. Wynik trafia na konsolę.

Oczywiście wiadomość musi być wcześniej stworzona z użyciem certyfikatu zawierającego klucz publiczny powiązany z używanym do jej odszyfrowania kluczem prywatnym.

Sposoby podawania kluczy w linii poleceń

Wiele poleceń wymaga podania klucza szyfrującego lub hasła. Można to zrobić interaktywnie (nie podajemy klucza w linii polecenia i program prosi wtedy o jego wpisanie), lub podać w linii polecenia używając następującej składni:

- **pass:<hasło>** - oznacza, że kluczem czy hasłem jest ciąg <hasło>.
- **env:<zmienna_systemowa>** - oznacza, że kluczem czy hasłem jest zawartość podanej zmiennej systemowej.
- **file:<plik>** - oznacza, że klucz czy hasło należy odczytać z podanego pliku.
- **stdin** - oznacza, że klucz czy hasło należy odczytać ze **stdin**, czyli konsoli.

Na przykład:

```
openssl enc -in tekst.txt -out tekst.enc -des -pass pass:1234
```

Szyfruje zawartość pliku **tekst.txt** używając algorytmu **DES** i hasła **1234**, a następnie zapisuje wynik do pliku **tekst.enc**.

```
openssl enc -in tekst.txt -out tekst.enc -3des -pass file:plik_z_haslem.txt
```

Szyfruje zawartość pliku **tekst.txt** używając algorytmu **3DES** i hasła odczytane z pliku **plik_z_haslem.txt**, a następnie zapisuje wynik do pliku **tekst.enc**.

Struktura folderów/plików CA opartego na pakiecie OpenSSL

W najprostszej wersji struktura katalogów i plików CA to:

- **newcerts** – (folder) zawiera pliki z wystawionymi certyfikatami. Pliki mają nazwy w postaci **<numer_seryjny>.pem**. Dodatkowe informacje o powyższych certyfikatach można znaleźć w pliku **index.txt**.
- **private** – (folder) zawiera dane poufne.
 - **CA.key** – klucz prywatny CA.
- **serial** – (plik) zawiera numer seryjny (2 cyfry) następnego certyfikatu, który będzie wystawiany.

- **index.txt** – (plik) zawiera dodatkowe informacje o wystawionych certyfikatach, odwołuje się do plików w folderze **newcerts**.
- **crlnumber** – (plik) zawiera numer seryjny (2 cyfry) ostatnio wygenerowanej listy CRL. Potrzebne tylko, jeśli zamierzamy generować listy CRL.

Przy podpisywaniu nowego certyfikatu:

1. Sprawdzana jest polityka (policy) danego CA. Jeśli żądanie nie spełnia wymagań, jest odrzucane.
2. Właściwości (np. czas ważności, możliwości zastosowania) zawarte w żądaniu są przepisywane do tworzonego certyfikatu. Można wyłączyć to działanie – wtedy wyłącznie CA decyduje o zawartych w certyfikacie właściwościach.
3. Jeśli zdefiniowaliśmy na poziomie CA (w pliku konfiguracyjnym lub z użyciem opcji linii polecenia) wartości jakiś właściwości, to są one umieszczane w certyfikacie. Jeśli występuje konflikt wartości żądanej przez użytkownika z właściwością ustawioną w CA (np. dotyczący czasu ważności), to wartości ustawione w CA zastępują te żądane przez użytkownika.
4. CA dodaje do certyfikatu informacje o sobie i podpisuje go swoim kluczem prywatnym.

Po podpisaniu nowego certyfikatu:

1. Pobierana jest wartość numeru seryjnego z pliku **serial**. W [**newcerts**] tworzony jest plik o nazwie **<numer_seryjny>.pem**, zawierający kopię wystawionego certyfikatu.
2. Do pliku **index.txt** dopisywana jest nowa linia zawierająca numer seryjny wystawionego certyfikatu (pobrany z pliku **serial**) i dodatkowe informacje o nim.
3. Plik z certyfikatem zapisywany w lokalizacji którą określiliśmy używając opcji **-out <plik>**.
4. Wartość w pliku **serial** jest zwiększana o 1.

Plik konfiguracyjny (openssl.cnf)

Plik konfiguracyjny ma domyślnie nazwę **openssl.cnf** i w przypadku dystrybucji Fedora Core znajduje się w folderze **/etc/pki/tls/**.

Polecenia korzystające z tego pliku (np. req i ca) pozwalają też na ręczne określenie jego lokalizacji przy użyciu opcji: **-config <plik>**

Plik konfiguracyjny zawiera 2 rodzaje informacji:

- wymagane do poprawnego działania polecenia pakietu OpenSSL– niemożliwe do podania żadnym innym sposobem (np. struktura katalogów CA),
- domyślne wartości parametrów, które inaczej musielibyśmy podawać z linii polecenia (np. domyślna nazwa pliku do którego zostanie zapisany klucz prywatny czy domyślna jego długość w bitach).

Używany podczas ćwiczenia plik konfiguracyjny zawiera następujące części:

- informacje wstępne
- informacje używane przez polecenie ca
- informacje używane przez polecenia req
- ogólne definicje, do których odwołują się pozostałe części.

Poniżej przedstawiono przykład pliku konfiguracyjnego, opatrzonych komentarzami – prosimy o przeanalizowanie jego struktury, a w szczególności:

- **struktury katalogów CA,**
- **definicji rozszerzeń dla certyfikatów typu self-signed,**

- polityk podpisywania certyfikatów przez CA,
- definicji rozszerzeń dla certyfikatów podpisywanych przez CA.

Przykład pliku konfiguracyjnego zawierający opis poszczególnych parametrów

```
#####
# Czesc dla polecenia ca #####
#####

[ ca ]
default_ca      = CA_default          # Wybor domyslnie uzywanego CA
                                           # przez odwołanie do sekcji ponizej
#####

[ CA_default ]

dir              = /etc/pki/przykladCA   # Główny folder CA
database         = $dir/index.txt       # Plik zawierający dodatkowe informacje o wyst. cert.
new_certs_dir    = $dir/newcerts        # Folder w którym zapis. sa kopie wystawionych cert.
certificate       = $dir/CA.cer         # CERTYFIKAT CA
serial           = $dir/serial          # Plik zaw. numer seryjny następnego certyfikatu
crlnumber        = $dir/crlnumber       # Plik zaw. numer seryjny aktualnej list CRL
crl              = $dir/crl.pem         # Aktualna lista CRL
private_key      = $dir/private/CA.key  # KLUCZ PRYWATNY CA
RANDFILE         = $dir/private/.rand   # Plik zawierający dane losowe

x509_extensions = usr_cert              # Właściwości dodawane do certyfikatu
name_opt         = ca_default           # Subject Name options
cert_opt         = ca_default           # Certificate field options

unique_subject   = no                   # Ustawienie na 'no' pozwala tworzyć
                                           # kilka cert. z takimi samymi tematami.

# Ustawienie na copy nakazuje kopiować do cert. właściwości z żądania
# Jeśli nie jest ustawione - właściwości ustawia tylko CA
# copy_extensions = copy

#####

# Właściwości dopisywane do listy CRL
crl_extensions   = crl_ext

default_days     = 365                   # jak długo ważna jest lista CRL
default_crl_days= 30                    # kiedy będzie dostępna nowa lista CRL
default_md       = sha1                  # funkcja skrotu używana przy podpisywaniu listy
preserve         = no                     # keep passed DN ordering

#####

# Wybor polityki podpisywania certyfikatow
policy           = policy_match

# Definicje polityk podpisywania certyfikatow
[ policy_match ]
countryName      = match                  # musi być takie same jak ma CA
stateOrProvinceName = match
organizationName = match
organizationalUnitName = optional        # nie musi być podane
commonName       = supplied              # musi być podane
emailAddress     = optional

[ policy_anything ]
countryName      = optional
stateOrProvinceName = optional
localityName     = optional
organizationName = optional
organizationalUnitName = optional
commonName       = supplied
emailAddress     = optional
```

```

#####
# Czesc dla polecenia req #####
#####

[ req ]
default_bits          = 1024                # domyslna dlugosc klucza prywatnego
default_md            = sha1                # stosowana funkcja skrotu
default_keyfile       = privkey.key        # domyslny plik do kt. trafi klucz prywatny
distinguished_name   = req_distinguished_name # nazwa definicji informacji identyfikacyjnych
attributes           = req_attributes      # nazwa definicji dodatkowych informacji do
                                                # umieszczenia w zadaniu certyfikatu
x509_extensions     = v3_selfsigned        # Wlasciwosci dodawane do certyfikatu typu self-signed
                                                # Ustawiane w czesci opisujacej zadanie, bo zadne CA
                                                # go nie podpisuje, tylko od razu klient swoim
                                                # kluczem prywatnym
req_extensions       = v3_request          # Propozycje rozszerz. dodawane do zadanania certyfikatu
                                                # wysylanego do CA
string_mask          = MASK:0x2002

#####

# definicja informacji identyfikacyjnych
# nazwy parametrow sa tu okreslone przez standard
[ req_distinguished_name ]
countryName           = Country Name (2 letter code)
countryName_default  = PL
countryName_min       = 2
countryName_max       = 2
stateOrProvinceName  = State or Province Name (full name)
stateOrProvinceName_default = Pomorskie
localityName          = Locality Name (eg, city)
localityName_default = Gdansk
0.organizationName   = Organization Name (eg, company)
0.organizationName_default = KG
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = Gdansk
commonName            = Common Name (eg, your name or your server\'s hostname)
commonName_max        = 64
emailAddress          = Email Address
emailAddress_max      = 64

#definicja dodolnych dodatkowych informacji - format jak wyzej
[ req_attributes ]
#przykladowyParametr = Tresc zapytania o wartosc parametru
#przykladowyParametr_min = 4 #minimalna dlugosc
#przykladowyParametr_max = 50 #maksymalna dlugosc
#przykladowyParametr_default = Domyslna wartosc parametru

#####
# Decinicje ogole, uzywane przez czesci powyzej #####
#####

# odwołanie w czesci ca - definicje wlasciwosci dodawanych przez CA do certyfikatu

[ usr_cert ]
basicConstraints=CA:FALSE          # cert nie moze byc uzyty do stworzenia CA
                                    # gdyz zakazujemy uzycia go do podpisywani cert.

subjectKeyIdentifier=hash          # informacje identyfikujace certyfikat/CA
authorityKeyIdentifier=keyid,issuer

# mozliwosci uzycia certyfikatu - standardowy zestaw dla cert. klienckiego
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = 1.3.6.1.5.5.7.3.2 # clientAuthentication

[ ca_cert ]
basicConstraints=CA:TRUE          # cert nie moze byc uzyty do stworzenia CA
                                    # gdyz zakazujemy uzycia go do podpisywani cert.

subjectKeyIdentifier=hash          # informacje identyfikujace certyfikat/CA
authorityKeyIdentifier=keyid,issuer

#####

# odwołanie w czesci req - wlasciwosci dodawane do zadanania certyfikatu
[ v3_request ]

```

```

basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# odwołanie w czesci req - wlasciwosci dodawane do certyfikatu typu self-signed
[ v3_selfsigned ]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer:always
basicConstraints = CA:true

#####

# odwołanie w czesci ca - wlasciwosci dopisywane do tworzonych list CRL
[ crl_ext ]
# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.

issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always,issuer:always

#####
#####
#####

```

Zarządzanie certyfikatami (Windows XP)

Zarządzanie certyfikatami w systemie Windows XP odbywa się z użyciem konsoli Microsoft Management Console. Predefiniowana konsola umożliwiająca zarządzanie certyfikatami aktualnego użytkownika może być uruchomiona z użyciem polecenia Start->Uruchom: **certmgr.msc**

Pojawiające się okno posiada 2 części: drzewo magazynów certyfikatów (po lewej) i widok zawartości aktualnie wybranego elementu drzewa (po prawej).

Magazyny certyfikatów przechowują certyfikaty różnych typów – nas interesują:

- **Osobisty** – tu znajdują się certyfikaty i klucze prywatne jakie posiada nasz użytkownik.
- **Zaufane główne urzędy certyfikacji** – zawiera certyfikaty typu self-signed urzędów certyfikacji typu root-CA (znajdujących się na szczycie hierarchii urzędów certyfikacji), które uznajemy za godne zaufania. Umieszczenie certyfikatu jakiegoś CA w tym magazynie, oznacza, że przy weryfikacji łańcucha certyfikatów (patrz „Sprawdzanie poprawności certyfikatu”) akceptujemy wszystkie łańcuchy zaczynające kończące się na danym CA.
- **Pośrednie urzędy certyfikacji** – zawiera certyfikaty pośrednich urzędów certyfikacji (intermediate). Służą one do tworzenia łańcucha certyfikatów przy sprawdzaniu poprawności certyfikatu.
- **Inne osoby** – certyfikaty innych użytkowników, które możemy wykorzystać, np. do wysyłania im zaszyfrowanych wiadomości.

W przypadku sprawdzania poprawności certyfikatu (patrz rozdziały poprzednie) w systemie Windows XP, łańcuch certyfikatów CA jest automatycznie tworzony z użyciem zawartości magazynu **Pośrednie urzędy certyfikacji**. Łańcuch taki może też być od razu dołączony do certyfikatu sprawdzanego przez jego właściciela (niektóre aplikacje udostępniają taką informację – np. serwer Apache jest w stanie udostępnić klientowi własny certyfikat, oraz cały, gotowy łańcuch aż do kończącego go root-CA).

Ostatni certyfikat łańcucha powinien znajdować się w magazynie **Główne zaufane urzędy certyfikacji** – jeśli tam będzie, to znaczy, że ufamy takiemu root-CA i w przypadku nieprzerwanego łańcucha możemy też zaufać certyfikatowi badanemu. Jeśli certyfikatu root-CA tam nie będzie, to pomimo poprawności ciągu, system nie zaakceptuje certyfikatu, gdyż jego autentyczność poświadcza niezaufane CA.