

PUBLIC KEY INFRASTRUCTURE

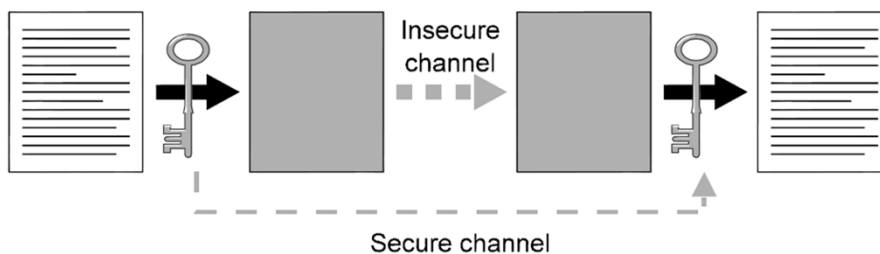
Elementy zaczerpnięte z:
Encryption & Security Tutorial (Peter Gutmann)
<http://www.cs.auckland.ac.nz/~pgut001/tutorial/>

Składowe bezpieczeństwa

- ▣ **Poufność** – uniemożliwia odczytanie przesyłanych informacji bez odpowiedniego klucza,
- ▣ **Integralność** – umożliwia stwierdzenie, czy odebrane dane nie uległy modyfikacji,
- ▣ **Uwierzytelnianie** – pozwala na weryfikację tożsamości,
- ▣ **Niezaprzeczalność** – pozwala na udowodnienie, że określona wiadomość (nie)została wysłana przez określonego nadawcę,
- ▣ **Dostępność** – gwarantuje nieprzerwany dostęp do zasobu,
- ▣ **Kontrola dostępu** – pozwala na kontrolę dostępu do zasobu.

Conventional Encryption

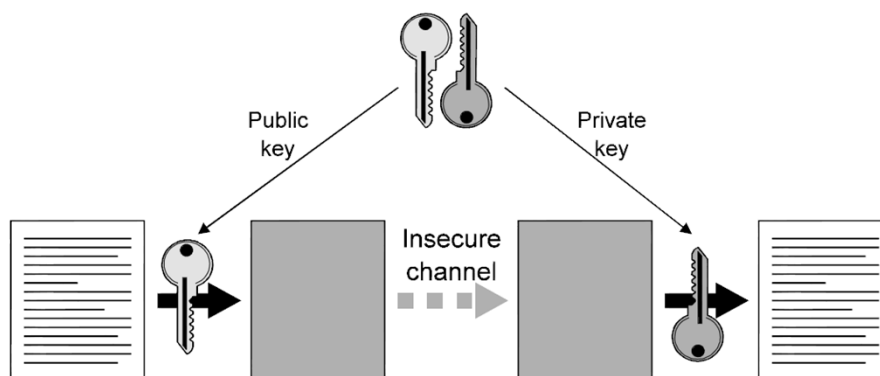
Uses a shared key



Problem of communicating a large message in secret is reduced to communicating a small key in secret

Public-key Encryption

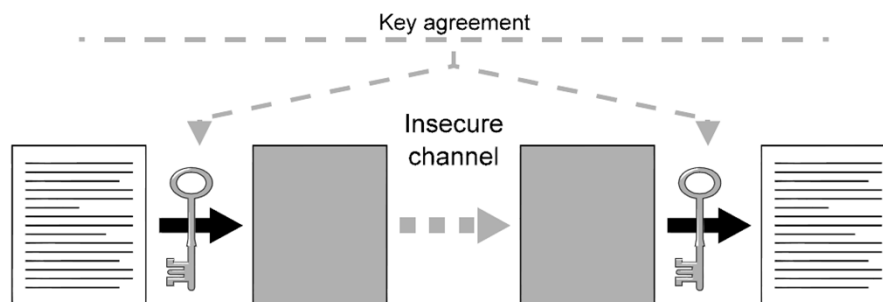
Uses matched public/private key pairs



Anyone can encrypt with the public key, only one person can decrypt with the private key

Key Agreement

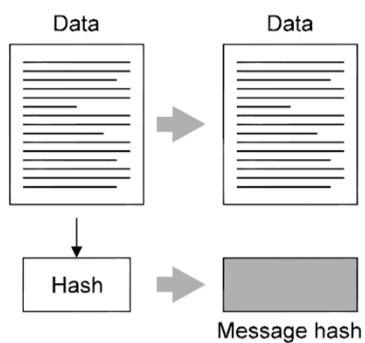
Allows two parties to agree on a shared key



Provides part of the required secure channel for exchanging a conventional encryption key

Hash Functions

Creates a unique “fingerprint” for a message

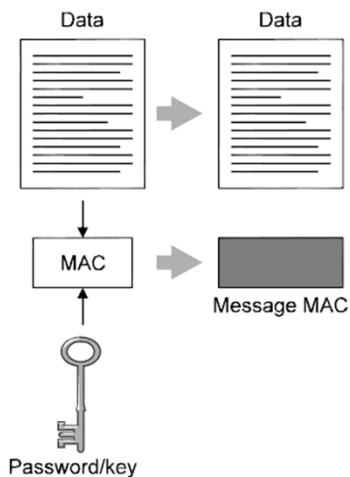


Anyone can alter the data and calculate a new hash value

- Hash has to be protected in some way

MAC's

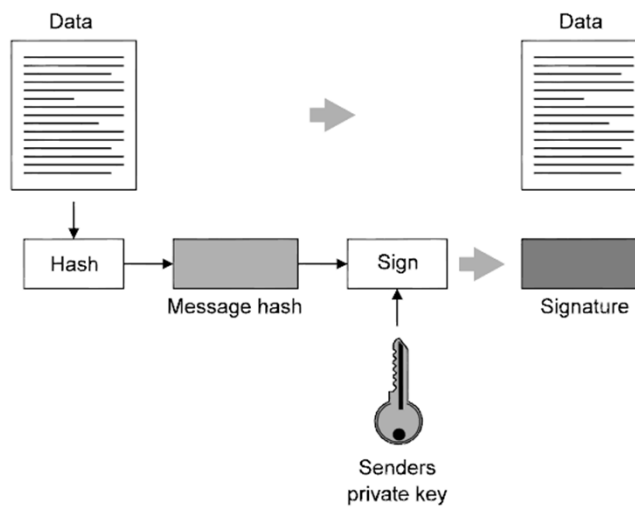
Message Authentication Code, adds a password/key to a hash



Only the password holder(s) can generate the MAC

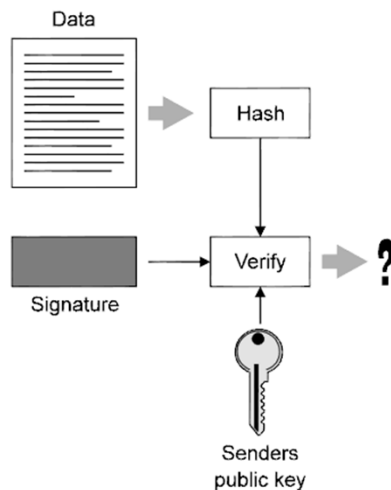
Digital Signatures

Combines a hash with a digital signature algorithm



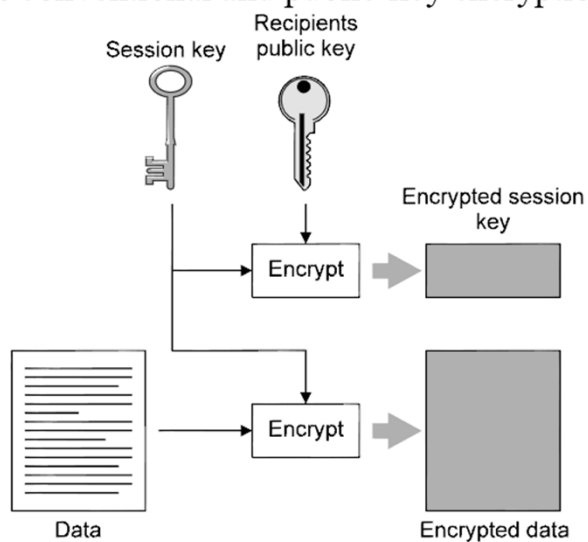
Digital Signatures (ctd)

Signature checking:

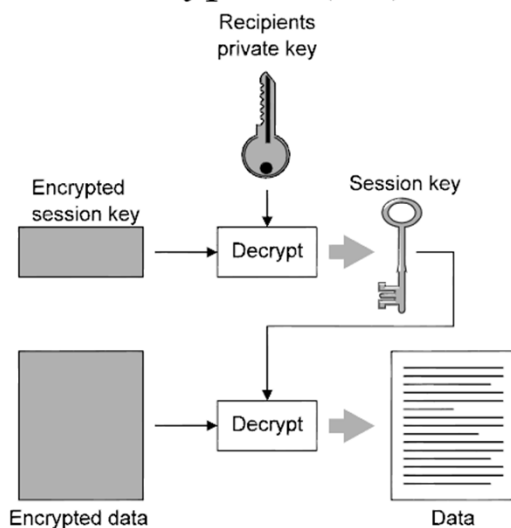


Message/Data Encryption

Combines conventional and public-key encryption



Message/data Encryption (ctd)



Public-key encryption provides a secure channel to exchange conventional encryption keys

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,
- ▣ **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.

RSA

- ▣ Wartości dane:
 - (n, e) = klucz publiczny, $n = p \cdot q$,
 - d = klucz prywatny,
 - M – wiadomość.
- ▣ Szyfrowanie: $C = M^e \bmod n$
- ▣ Rozszyfrowanie: $M = C^d \bmod n$
- ▣ Przykład:
 - $p, q = 5, 7, n = p \cdot q = 35, e = 5$; Klucz publiczny $(35, 5)$.
 - Klucz prywatny: $d = e^{-1} \bmod ((p-1)(q-1)) = 5$

RSA

- ▣ Wiadomość:
 - $M = 4$
- ▣ Szyfrowanie:
 - $C = 4^5 \bmod 35$
 - $C = 45^d \bmod 35 = 9$
- ▣ Rozszyfrowanie:
 - $M = 9$
 - $M = 95 \bmod 35 = 59049 \bmod 35 = 4$

DH

U1		U2		U3	
$p = 23$	$b = ?$	$p = 23$	$a = ?$	$p = 23$	$a = ?$
base $g = 5$		base $g = 5$		base $g = 5$	$b = ?$
$a = 6$		$b = 15$			$s = ?$
$A = 5^a \bmod 23 = 8$		$B = 5^{15} \bmod 23 = 19$		$A = 5^a \bmod 23 = 8$	
$B = 5^b \bmod 23 = 19$		$A = 5^a \bmod 23 = 8$		$B = 5^b \bmod 23 = 19$	
$s = 19^b \bmod 23 = 2$		$s = 8^{15} \bmod 23 = 2$		$s = 19^a \bmod 23$	
$s = 8^b \bmod 23 = 2$		$s = 19^a \bmod 23 = 2$		$s = 8^b \bmod 23$	
$s = 19^b \bmod 23 = 8^b \bmod 23$		$s = 8^{15} \bmod 23 = 19^a \bmod 23$		$s = 19^a \bmod 23 = 8^b \bmod 23$	
$s = 2$		$s = 2$			

- ▣ Oznaczenia:
 - p, g - wartości uzgodnione pomiędzy U1 i U2
 - a, b - wartości tajne wybrane przez U1 i U2
 - A, B - wartości przesyłane
 - s - wspólny, tajny klucz

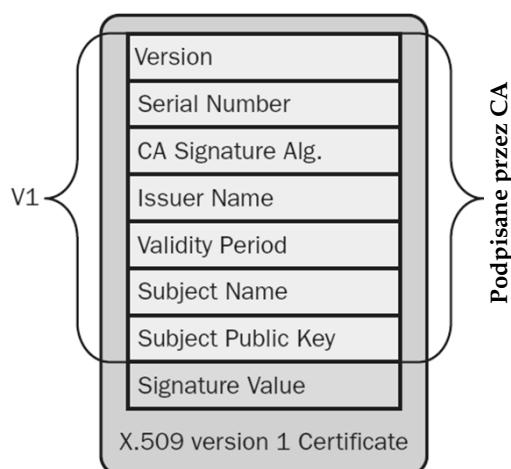
Długości klucza (oszacowanie)

Algorytm symetryczny	Algorytm asymetryczny	Algorytm ECC
40	-	-
56	400	-
64	512	-
80	768	-
90	1024	160
112	1792	195
120	2048	210
128	2304	256

- ▣ Dodatkowo: klucz symetryczny używany jest raz na wiadomość/sesję, a klucz asymetryczny wielokrotnie.

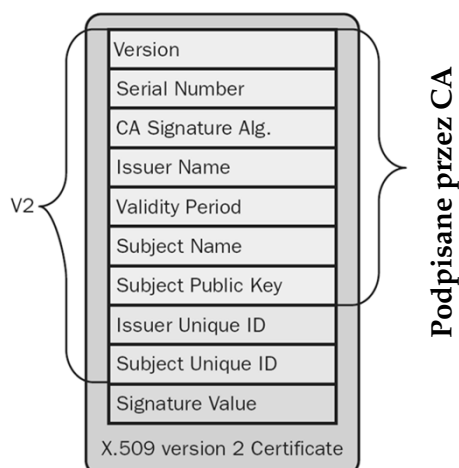
Certyfikaty X.509 v1

- ▣ Zawiera:
 - pola „techniczne”,
 - nazwę właściciela,
 - nazwę wystawcy,
 - okres ważności,
 - klucz publiczny właściciela.
- ▣ Podpisane przez wystawiające CA.



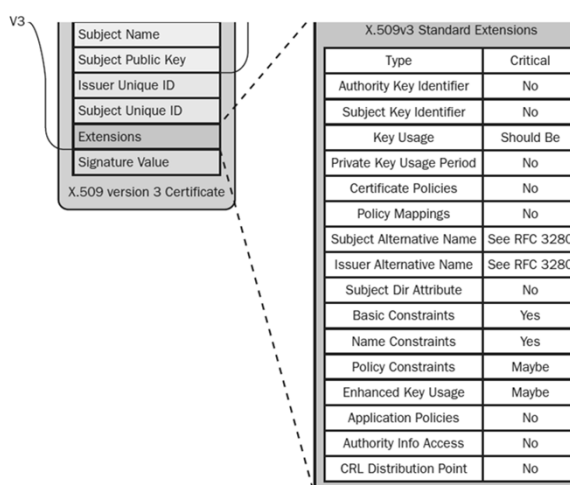
Certyfikaty X.509 v2

- Wprowadzono pola:
 - Issuer Unique ID
 - Subject Unique ID
- w celu:
 - zmniejszenia prawdopodobieństwa „kolizji nazw”,
 - poprawy działania mechanizmów tworzenia łańcuchów certyfikatów,
 - poprawy wsparcia dla odnawiania certyfikatów CA.
- Obecnie (RFC3280) użycie tych pól jest niezalecane.



Certyfikaty X.509 v3

- Wersja 3 stanowi rozbudowę wersji 2 o tzw. rozszerzenia (extensions).
- Rozszerzenia identyfikowane są z użyciem OID.
- Rozszerzenia oznaczone jako krytyczne (critical) muszą zostać rozpoznane przez aplikację, albo certyfikat nie zostanie użyty.



Ważniejsze rozszerzenia

- **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.

Ważniejsze rozszerzenia

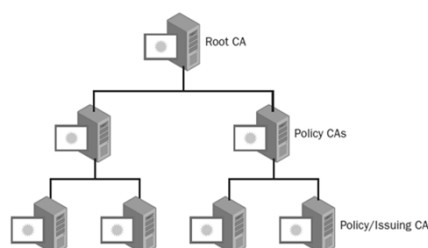
- **Enhanced key usage** – zawiera bardziej szczegółowe informacje o możliwości wykorzystania certyfikatu. Mają one postać listy identyfikatorów OID oznaczających różne zastosowania. Aplikacja może wymagać obecności określonych OID, aby wzięła pod uwagę dany certyfikat. Przykłady najpopularniejszych:
 - 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.

Ważniejsze rozszerzenia

- ▣ **Basic constraints** – pozwala na określenie czy certyfikat przeznaczony jest dla CA, użytkownika, komputera, urządzenia sieciowego...
Pozwala także na określenie maksymalnej liczby poziomów potomnych CA.
- ▣ **Name constraints** – określa dozwolone nazwy podmiotów w wystawianych przez CA certyfikatach.

Certificate Authority

- ▣ Zadania CA:
 - Weryfikuje podmiot
 - Wystawia certyfikat
 - Zarządza odwołaniami
- ▣ Podział:
 - Root/Intermediate
 - Parent/child
 - Offline/Online
 - Root, Policy, Issuing



Proces uzyskiwania certyfikatu

- ▣ Klient generuje klucze publiczny i prywatny.
- ▣ 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.
- ▣ Klient tworzy żądanie certyfikatu (Certificate Signing Request - CSR),, łącząc:
 - klucz publiczny,
 - dodatkowe informacje identyfikacyjne, np. nazwę, kraj, miasto (najważniejszym parametrem jest **common name**)
 - pożądane rozszerzenia certyfikatu, np.: okres ważności, zbiór możliwych sposobów wykorzystania certyfikatu itp.

Proces uzyskiwania certyfikatu

- ▣ Żądanie certyfikatu przesyłane jest do urzędu certyfikacji (CA), które:
 - sprawdza techniczną poprawność (format) żądania,
 - 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,
 - sprawdza czy żądane przez klienta rozszerzenia certyfikatu są zgodne z dopuszczalnymi:
 - część rozszerzeń z żądania może zostać umieszczona w certyfikacie bez zmian,
 - część może zostać usunięta/zmodyfikowana, a nowe rozszerzenia dopisane do generowanego certyfikatu.
 - klucz publiczny wraz z dodatkowymi informacjami i rozszerzeniami zostaje podpisany przez CA jego kluczem prywatnym, tworząc certyfikat.
- ▣ Klient otrzymuje certyfikat, który jest potem wykorzystywany w połączeniu z jego kluczem prywatnym.

Listy 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.
- ▣ 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.
- ▣ W przypadku niektórych funkcji, nowe (2008+, Win7) systemy operacyjne MS sprawdzają CRL przy każdym użyciu danego certyfikatu. Brak dostępu do listy CRL traktowany jest jako stan krytyczny, uniemożliwiający użycie certyfikatu.
 - Dotyczy to również braku informacji o sposobie dostępu do danego CRL.

Listy CRL

- ▣ 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. Jedyne przypadki, gdy certyfikat może odzyskać ważność.

Weryfikacja certyfikatu

- Otrzymany certyfikat sprawdzany jest pod względem podstawowej poprawności formatu.
- 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).
- Dla całego łańcucha:
 - Sprawdzane są krytyczne rozszerzenia certyfikatu (własności które muszą być rozumiane przez aplikację) – jeśli występuje jakaś niezrozumiała, certyfikat jest odrzucony.
 - Sprawdzane są rozszerzenia certyfikatu, aby sprawdzić, czy jest on możliwy do życia w roli w której próbuje go wykorzystać aplikacja (np. do podpisywania).
 - Sprawdzane są listy CRL, w celu weryfikacji, czy żaden z certyfikatów nie został odwołany.
 - Sprawdzana jest poprawność podpisów CA, w celu wykrycia ewentualnych, nieuprawnionych modyfikacji certyfikatów.
 - Sprawdzany jest okres ważności certyfikatu.
- **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.**

Public Key Cryptography Standards

	Name	Comments
PKCS #1	RSA Cryptography Standard	See RFC 3447. Defines the mathematical properties and format of RSA public and private keys (ASN.1-encoded in clear-text), and the basic algorithms and encoding/padding schemes for performing RSA encryption, decryption, and producing and verifying signatures.
PKCS #3	Diffie-Hellman Key Agreement Standard	A cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel.
PKCS #5	Password-based Encryption Standard	See RFC 2898 and PBKDF2.
PKCS #7	Cryptographic Message Syntax Standard	See RFC 2315. Used to sign and/or encrypt messages under a PKI. Used also for certificate dissemination (for instance as a response to a PKCS#10 message). Formed the basis for S/MIME, which is as of 2010 based on RFC 5652, an updated Cryptographic Message Syntax Standard (CMS). Often used for single sign-on.
PKCS #8	Private-Key Information Syntax Standard	See RFC 5208. Used to carry private certificate keypairs (encrypted or unencrypted).
PKCS #9	Selected Attribute Types	Defines selected attribute types for use in PKCS #6 extended certificates, PKCS #7 digitally signed messages, PKCS #8 private-key information, and PKCS #10 certificate-signing requests.
PKCS #10	Certification Request Standard	See RFC 2986. Format of messages sent to a certification authority to request certification of a public key. See certificate signing request.
PKCS #11	Cryptographic Token Interface (Cryptoki)	An API defining a generic interface to cryptographic tokens (see also Hardware Security Module). Often used in single sign-on, Public-key cryptography and disk encryption systems.
PKCS #12	Personal Information Exchange Syntax Standard	Defines a file format commonly used to store private keys with accompanying public key certificates, protected with a password-based symmetric key. PFX is a predecessor to PKCS#12. This container format can contain multiple embedded objects, such as multiple certificates. Usually protected/encrypted with a password. Usable as a format for the Java key store. Usable by Tomcat, but not by Apache.
PKCS #13	Elliptic Curve Cryptography Standard	
PKCS #14	Pseudo-random Number Generation	
PKCS #15	Cryptographic Token Information Format Standard	Defines a standard allowing users of cryptographic tokens to identify themselves to applications, independent of the application's Cryptoki implementation (PKCS #11) or other API.

OpenSSL

OpenSSL

Standard commands

asn1parse	ca	ciphers	cr1
cr12pkcs7	dgst	dh	dhparam
dsa	dsaparam	enc	engine
errstr	gendh	gensa	genrsa
nseq	ocsp	passwd	pkcs12
pkcs7	pkcs8	prime	rand
req	rsa	rsautl	s_client
s_server	s_time	sess_id	smime
speed	spkac	verify	version
x509			

Message Digest commands

md2	md4	md5	rmd160
sha	sha1		

Cipher commands

aes-128-cbc	aes-128-ecb	aes-192-cbc	aes-192-ecb
aes-256-cbc	aes-256-ecb	base64	bf
bf-cbc	bf-cfb	bf-ecb	bf-ofb
camellia-128-cbc	camellia-128-ecb	camellia-192-cbc	camellia-192-ecb
camellia-256-cbc	camellia-256-ecb	cast	cast-cbc
cast5-cbc	cast5-cfb	cast5-ecb	cast5-ofb
des	des-cbc	des-cfb	des-ecb
des-ede	des-ede-cbc	des-ede-cfb	des-ede-ofb
des-ede3	des-ede3-cbc	des-ede3-cfb	des-ede3-ofb
des-ofb	des3	desx	rc2
rc2-40-cbc	rc2-64-cbc	rc2-cbc	rc2-cfb
rc2-ecb	rc2-ofb	rc4	rc4-40
seed	seed-cbc	seed-cfb	seed-ecb
seed-ofb			

Wybrane polecenia

- ▣ **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.

Plik konfiguracyjny: openssl.cnf

Openssl.cnf

- ▣ Używany przez polecenia:
 - ca – działania związane z lokalnym CA,
 - req – generowanie żądań certyfikatów,
 - x509 – działania związane z wystawionymi już certyfikatami.
- ▣ Odczytywany jest z folderu ustalonego podczas kompilacji pakietu OpenSSL
 - Możliwe podanie ścieżki z linii polecenia.
- ▣ Podzielony na sekcje o tytułach umieszczonych w [].

Sekcja [ca]

- ▣ Pozwala na obsługę kilku niezależnych CA
- ▣ Zawiera odwołanie do właściwej sekcji CA
 - default_ca = ...

Domyślna sekcja [CA_default]

- ▣ Stanowi podstawę definicji CA.
- ▣ Dane podstawowe:
 - **dir** = /etc/pki/przykladCA - główny folder CA. Używany dalej jako **\$dir**
 - **certificate** = \$dir/CA.cer - certyfikat CA,
 - **private_key** = \$dir/private/CA.key - klucz prywatny CA,
 - **RANDFILE** = \$dir/private/rand - plik służący generacji danych losowych,
 - **policy** = **policy_match** - zawiera odwołanie do sekcji określającej politykę przyznawania certyfikatów.
- ▣ Baza wystawionych certyfikatów:
 - **database** = \$dir/index.txt - Plik zawierający informacje o wystawionych certyfikatach,
 - **new_certs_dir** = \$dir/newcerts - Folder w którym zapisywane są kopie wystawionych certyfikatów,
 - **serial** = \$dir/serial - Plik zawiera numer seryjny następnego certyfikatu.

Domyślna sekcja [CA_default]

- ▣ Lista CRL:
 - **crlnumber** = \$dir/crlnumber - zawiera numer seryjny aktualnej listy CRL,
 - **crl** = \$dir/crl.pem - plik zawierający aktualną listę CRL.
 - **default_days** = 365 - jak długo ważna jest lista CRL
 - **default_crl_days** = 30 - kiedy będzie dostępna nowa lista CRL
 - **default_md** = sha1 - funkcja skrótu używana przy podpisywaniu listy CRL
 - **crl_extensions** = **crl_ext** - zawiera odwołanie do sekcji pozwalającej zdefiniować rozszerzenia listy CRL.
- ▣ Parametry wystawianych certyfikatów:
 - **x509_extensions** = **usr_cert** - właściwości dodawane do wystawianego certyfikatu przez CA.
 - **unique_subject** = **no** - Ustawienie na 'no' pozwala tworzyć kilka certyfikatów z takimi samymi tematami.
 - **copy_extensions** = **copy** - Ustawienie na copy nakazuje kopiować do cert. właściwości z żądania. Jeśli nie jest ustawione - właściwości ustawia tylko CA

Domyślna sekcja [policy_match]

- ▣ Określa jakie warunki musi spełniać żądanie certyfikatu, aby zostało podpisane.
 - Zawiera pola informacyjne certyfikatu oraz określenia:
 - ▣ match – pole musi mieć taką samą wartość, jak analogiczne pole certyfikatu CA,
 - ▣ supplied – pole musi być obecne i zawierać dane,
 - ▣ optional – pole nie jest wymagane.
 - Np.: countryName = match

Domyślna sekcja [req]

- ▣ Zawiera domyślne parametry generowanych żądań certyfikatów.
 - **default_bits = 1024** - domyślna długość klucza prywatnego,
 - **default_md = sha1** - funkcja skrótu stosowana przy podpisywaniu certyfikatu,
 - **default_keyfile = privkey.key** - domyślny plik do którego trafi klucz prywatny,
 - **distinguished_name = req_distinguished_name** - nazwa sekcji definiującej informacje o właścicielu zawarte w żądaniu,
 - **attributes = req_attributes** - nazwa definicji dodatkowych informacji do umieszczenia w żądaniu certyfikatu,
 - **x509_extensions = v3_selfsigned** - właściwości dodawane do certyfikatu typu self-signed.
 - **req_extensions = v3_request** - Propozycje rozszerzeń dodawane do żądania certyfikatu.

Domyślna sekcje [req_distinguished_name] i [req_attributes]

- ▣ Zawiera definicje pól do umieszczenia w żądaniu certyfikatu, wraz z dodatkowymi parametrami.
- ▣ Format:
 - przykładowyParametr = Treść zapytania o wartość parametru
 - przykładowyParametr_min = 4 - minimalna długość pola
 - przykładowyParametr_max = 50 - maksymalna długość pola
 - przykładowyParametr_default = Domyślna wartość parametru
- ▣ Np.:
 - countryName = Country Name (2 letter code)
 - countryName_default = PL
 - countryName_min = 2
 - countryName_max = 2

Domyślne sekcje [v3_request] i [v3_selfsigned]

- ▣ Zawiera domyślne właściwości zawarte w generowanym:
 - żądaniu certyfikatu – [v3_request]
 - certyfikacie samo-podpisanym – [v3_selfsigned]
- ▣ Np:
 - basicConstraints = CA:FALSE
 - keyUsage = nonRepudiation, digitalSignature, keyEncipherment

Domyślna sekcja [usr_cert]

- ▣ Określa właściwości certyfikatu wystawionego przez CA.
- ▣ Np:
 - **basicConstraints=CA:FALSE** - cert nie może być użyty do stworzenia CA.
 - **subjectKeyIdentifier=hash** - informacje identyfikujące certyfikat
 - **authorityKeyIdentifier=keyid,issuer** – informacje identyfikujące CA
 - **keyUsage = nonRepudiation, digitalSignature, keyEncipherment** – dopuszczalne użycie certyfikatu
 - **extendedKeyUsage = 1.3.6.1.5.5.7.3.1** - szczegółowe przeznaczenie certyfikatu (serverAuth)
 - **subjectAltName = DNS:<dns_name>,DNS:<IP_addr>** - alternatywne nazwy właściciela certyfikatu.
 - **crlDistributionPoints=URI:http://crl.pg.gda.pl/pg.crl** – lokalizacja listy CRL