Generowanie certyfikatów: OpenSSL i urząd certyfikacji Windows

Poziom: średni

Dotyczy: OpenSSL 3.x, Windows Server 2022 (AD CS, urząd CA-Zielona na dc01)

Czas: ~35 min

Ta strona pokazuje, jak od zera wygenerować klucz i żądanie certyfikatu, jak dostać podpisany certyfikat od naszego urzędu CA-Zielona na Windowsie i jak zbudować własną małą CA do testów. Podgląd, konwersje formatów i certbot opisujemy na stronie Certyfikaty SSL: wystawienie, konwersja, weryfikacja, tutaj do niej wracamy tylko linkiem.

Skróty użyte na tej stronie (16)

Skrót	Co znaczy
AD CS	Active Directory Certificate Services: rola Windows Server pełniąca funkcję firmowego urzędu certyfikacji
AIA	Authority Information Access: miejsce w certyfikacie, które mówi, skąd pobrać certyfikat urzędu, który go podpisał
CA	Certificate Authority: urząd certyfikacji, który podpisuje certyfikaty
CDP	CRL Distribution Point: adres w certyfikacie, pod którym urząd publikuje listę unieważnionych certyfikatów
CRL	Certificate Revocation List: lista certyfikatów unieważnionych przed upływem ich terminu ważności
CSR	Certificate Signing Request: wniosek o wystawienie certyfikatu
DER	binarny, nieczytelny dla człowieka format zapisu certyfikatu
EC	Elliptic Curve: kryptografia na krzywych eliptycznych, krótsze klucze przy podobnym bezpieczeństwie jak RSA
ECDSA	Elliptic Curve Digital Signature Algorithm: podpis cyfrowy oparty na krzywych eliptycznych

Skrót	Co znaczy
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
JKS	Java KeyStore: magazyn certyfikatów i kluczy używany przez aplikacje Javy
PEM	Privacy-Enhanced Mail: tekstowy format certyfikatu zaczynający się od -----BEGIN
PFX	plik PKCS#12 z certyfikatem i kluczem prywatnym razem, typowy w Windowsie
RSA	Rivest, Shamir, Adleman: algorytm szyfrowania kluczem publicznym, którym Vaultwarden podpisuje tokeny sesji, stąd pliki rsa_key* w danych aplikacji
SAN	w certyfikatach: Subject Alternative Name, dodatkowe nazwy hosta w certyfikacie; w pamięciach: Storage Area Network, sieć dyskowa
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS

Fiszka

Komendy `openssl` wykonujemy w bash na `docker01` albo na stacji administratora, komendy `certreq` i `certutil` w PowerShellu uruchomionym jako administrator na serwerze Windows. Pojęcia CSR, SAN, PEM i PFX mamy wyjaśnione w Certyfikaty SSL: wystawienie, konwersja, weryfikacja.

□□ PLIK KONFIGURACYJNY

Plik konfiguracyjny żądania: san.cnf

```
[req]
default_bits       = 3072
default_md         = sha256
prompt             = no
distinguished_name = dn
req_extensions     = v3_req

[dn]
C  = PL
ST = Wielkopolskie
L  = Pamiatkowo
O  = Zielona sp. z o.o.
OU = IT
CN = git.zielona.example
```

```
[v3_req]
basicConstraints = CA:FALSE
keyUsage         = digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
subjectAltName   = @alt_names

[alt_names]
DNS.1 = git.zielona.example
DNS.2 = docker01.ad.zielona.example
```

Plik trzymamy obok klucza i podmieniamy nazwy na własne. Sekcja `[alt_names]` to lista nazw, pod którymi klienci łączą się z serwerem, a `prompt = no` mówi OpenSSL, żeby nie pytał o pola interaktywnie.

OU = Organizational Unit: folder w Active Directory grupujący konta i komputery · **IT** = Information Technology: dział albo praca związana z komputerami i infrastrukturą firmy · **CN** = Common Name: główna nazwa w certyfikacie, np. adres serwera · **CA** = Certificate Authority: urząd certyfikacji, który podpisuje certyfikaty · **DNS** = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP

ZMIENIA SYSTEM

Wygenerowanie klucza RSA i żądania z SAN: openssl req -config

```
openssl req -new -newkey rsa:3072 -nodes -keyout NazwaSerwera.key -out NazwaSerwera.csr
-config san.cnf
```

Jednym poleceniem powstaje klucz prywatny (3072 bity) i żądanie podpisania z nazwami z pliku `san.cnf`. W OpenSSL 3.x przełącznik `-nodes` ma nowszy odpowiednik `-noenc`, oba działają i oznaczają klucz bez hasła. Plik `.key` zostaje na serwerze i nigdy nie wędruje do urzędu CA.

Przykład:

```
.......+++++++++++++++++++++++++++++++++++++++*...++++++++++++++++++++++++++++++++++++
+*.....
-----
```

RSA = Rivest, Shamir, Adleman: algorytm szyfrowania kluczem publicznym, którym Vaultwarden podpisuje tokeny sesji, stąd pliki rsa_key* w danych aplikacji · **SAN** = w certyfikatach: Subject Alternative Name, dodatkowe nazwy hosta w certyfikacie; w pamięciach: Storage Area Network, sieć dyskowa · **CA** = Certificate Authority: urząd certyfikacji, który podpisuje certyfikaty · **openssl** =

zestaw narzędzi do certyfikatów i szyfrowania

□□ ZMIENIA SYSTEM

Wygenerowanie klucza EC i żądania: openssl ecparam

```
openssl ecparam -genkey -name prime256v1 -noout -out NazwaSerwera.key
openssl req -new -key NazwaSerwera.key -out NazwaSerwera.csr -subj "/C=PL/O=Zielona sp.
z o.o./CN=vpn.zielona.example" -addext "subjectAltName=DNS:vpn.zielona.example"
```

Klucz na krzywej eliptycznej P-256 jest krótszy i szybszy od RSA 3072 przy porównywalnym bezpieczeństwie. Zanim wyślemy takie żądanie do CA-Zielona, sprawdzamy, czy szablon dopuszcza klucze ECDSA, bo część szablonów dopuszcza wyłącznie RSA.

□□ **EC** = Elliptic Curve: kryptografia na krzywych eliptycznych, krótsze klucze przy podobnym bezpieczeństwie jak RSA · **RSA** = Rivest, Shamir, Adleman: algorytm szyfrowania kluczem publicznym, którym Vaultwarden podpisuje tokeny sesji, stąd pliki rsa_key* w danych aplikacji · **ECDSA** = Elliptic Curve Digital Signature Algorithm: podpis cyfrowy oparty na krzywych eliptycznych · **openssl** = zestaw narzędzi do certyfikatów i szyfrowania · **DNS** = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP

□□ TYLKO ODCZYT

Podgląd żądania: openssl req -text

```
openssl req -in NazwaSerwera.csr -noout -text | grep -E 'Subject:|Public-Key|DNS:'
```

Przykład:

```
        Subject: C = PL, ST = Wielkopolskie, L = Pamiatkowo, O = Zielona sp. z o.o., OU
= IT, CN = git.zielona.example
                Public-Key: (3072 bit)
                DNS:git.zielona.example, DNS:docker01.ad.zielona.example
```

□□ **openssl** = zestaw narzędzi do certyfikatów i szyfrowania · **DNS** = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP

□□ TYLKO ODCZYT

Sprawdzenie podpisu żądania: openssl req -verify

```
openssl req -in NazwaSerwera.csr -noout -verify
```

Przykład:

```
Certificate request self-signature verify OK
```

openssl = zestaw narzędzi do certyfikatów i szyfrowania

TYLKO ODCZYT

Podgląd certyfikatu w skrócie: openssl x509

```
openssl x509 -in NazwaSerwera.crt -noout -subject -issuer -dates -ext subjectAltName
```

Wypisujemy tylko cztery rzeczy, które najczęściej sprawdzamy po odebraniu certyfikatu: dla kogo, kto podpisał, do kiedy ważny i jakie ma nazwy. Pełny podgląd (`-text`) opisujemy w Certyfikaty SSL: wystawienie, konwersja, weryfikacja.

Przykład:

```
subject=C = PL, ST = Wielkopolskie, L = Pamiatkowo, O = Zielona sp. z o.o., OU = IT, CN
= git.zielona.example
issuer=DC = example, DC = zielona, DC = ad, CN = CA-Zielona
notBefore=Oct  3 08:12:00 2026 GMT
notAfter=Oct  3 08:22:00 2027 GMT
X509v3 Subject Alternative Name:
    DNS:git.zielona.example, DNS:docker01.ad.zielona.example
```

SSL = Secure Sockets Layer: dawna nazwa szyfrowania, dziś formalnie TLS · **openssl** = zestaw narzędzi do certyfikatów i szyfrowania

TYLKO ODCZYT

Weryfikacja łańcucha z certyfikatem pośrednim: openssl verify

```
openssl verify -CAfile NazwaKorzenia.crt -untrusted NazwaPosredniego.crt -purpose
sslserver NazwaSerwera.crt
```

(gdzie `-CAfile` wskazuje certyfikat główny, któremu ufamy, a `-untrusted` certyfikat pośredni, który tylko łączy serwer z korzeniem, ale sam z siebie nie jest dla nas wiarygodny)

Przykład:

```
NazwaSerwera.crt: OK
```

Gdy certyfikat pośredni brakuje, polecenie kończy się błędem `error 20 at 0 depth lookup: unable to get local issuer certificate`. Jeśli nasza CA ma tylko jeden poziom, jak CA-Zielona, wystarczy `-CAfile` z jej certyfikatem i bez `-untrusted`. Po przenosinach urzędu na `ca01` łańcuch ma dwa poziomy: w `-CAfile` stoi `Zielona-Root-CA`, a w `-untrusted` `Zielona-CA01`.

□□ **CA** = Certificate Authority: urząd certyfikacji, który podpisuje certyfikaty · **openssl** = zestaw narzędzi do certyfikatów i szyfrowania

□□ **TYLKO ODCZYT**

Sprawdzenie zgodności klucza z certyfikatem, także dla EC: skrót klucza publicznego

```
openssl x509 -in NazwaSerwera.crt -noout -pubkey | openssl sha256
openssl pkey -in NazwaSerwera.key -pubout | openssl sha256
```

Porównanie modulusów ze strony Certyfikaty SSL: wystawienie, konwersja, weryfikacja działa tylko dla RSA. Skrót klucza publicznego działa dla RSA i EC, bo porównujemy całą publiczną część klucza. Dwie identyczne linie oznaczają parę.

Przykład:

```
SHA2-256(stdin)= 3b1f0a6e8c24d97f5a02b4c1e7d83f60a9c5e2b17d4f8a30c6e91b5d27a4f08c
SHA2-256(stdin)= 3b1f0a6e8c24d97f5a02b4c1e7d83f60a9c5e2b17d4f8a30c6e91b5d27a4f08c
```

□□ **EC** = Elliptic Curve: kryptografia na krzywych eliptycznych, krótsze klucze przy podobnym bezpieczeństwie jak RSA · **SSL** = Secure Sockets Layer: dawna nazwa szyfrowania, dziś formalnie TLS · **RSA** = Rivest, Shamir, Adleman: algorytm szyfrowania kluczem publicznym, którym Vaultwarden podpisuje tokeny sesji, stąd pliki rsa_key* w danych aplikacji · **openssl** = zestaw narzędzi do certyfikatów i szyfrowania

□□ **ZMIENIA SYSTEM**

Sprawdzenie certyfikatu serwera na żywo: openssl s_client

```
openssl s_client -connect git.zielona.example:443 -servername git.zielona.example -
CAfile CA-Zielona.crt -verify_return_error </dev/null 2>&1 | grep -E
```

```
'Verification|Verify return code'
```

Łączymy się z serwerem tak, jak zrobiłaby to przeglądarka, i sprawdzamy, czy certyfikat przechodzi weryfikację względem naszego pliku `CA-Zielona.crt`. Parametr `-servername` jest potrzebny, gdy pod jednym adresem IP stoi kilka witryn. Wyświetlenie samych dat ważności opisujemy w Certyfikaty SSL: wystawienie, konwersja, weryfikacja.

Przykład:

```
Verification: OK
Verify return code: 0 (ok)
```

IP = Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection) · **SSL** = Secure Sockets Layer: dawna nazwa szyfrowania, dziś formalnie TLS · **openssl** = zestaw narzędzi do certyfikatów i szyfrowania

TYLKO ODCZYT

Rozpoznanie formatu pliku z CA: head

```
head -n 1 certnew.cer
```

Jeśli pierwsza linia to `-----BEGIN CERTIFICATE-----`, plik jest w PEM i OpenSSL czyta go od razu. Jeśli w terminalu pojawiają się krzaczki, plik jest w DER i do każdego polecenia `openssl x509` dodajemy `-inform der`.

CA = Certificate Authority: urząd certyfikacji, który podpisuje certyfikaty · **PEM** = Privacy-Enhanced Mail: tekstowy format certyfikatu zaczynający się od -----BEGIN · **DER** = binarny, nieczytelny dla człowieka format zapisu certyfikatu · **head** = pokaż początek pliku, domyślnie dziesięć pierwszych linii

ZMIENIA SYSTEM

Konwersja DER do PEM: openssl x509 -inform der

```
openssl x509 -inform der -in certnew.cer -out certnew.pem
```

Zestaw pozostałych konwersji (PEM do DER, PEM do PFX, PFX do PEM, PFX do JKS) mamy w tabeli na stronie Certyfikaty SSL: wystawienie, konwersja, weryfikacja.

DER = binarny, nieczytelny dla człowieka format zapisu certyfikatu · **PEM** = Privacy-Enhanced Mail: tekstowy format certyfikatu zaczynający się od -----BEGIN · **PFX** = plik PKCS#12 z certyfikatem i kluczem prywatnym razem, typowy w Windowsie · **JKS** = Java KeyStore: magazyn certyfikatów i

kluczy używany przez aplikacje Javy · **SSL** = Secure Sockets Layer: dawna nazwa szyfrowania, dziś formalnie TLS · **openssl** = zestaw narzędzi do certyfikatów i szyfrowania

PLIK KONFIGURACYJNY

Plik odpowiedzi dla certreq: request.inf

```
[Version]
Signature="$Windows NT$"

[NewRequest]
Subject = "CN=vpn.zielona.example, OU=IT, O=Zielona sp. z o.o., L=Pamiatkowo,
S=Wielkopolskie, C=PL"
KeyAlgorithm = RSA
KeyLength = 3072
HashAlgorithm = sha256
MachineKeySet = TRUE
Exportable = FALSE
RequestType = PKCS10

[EnhancedKeyUsageExtension]
OID = 1.3.6.1.5.5.7.3.1

[Extensions]
2.5.29.17 = "{text}"
_continue_ = "dns=vpn.zielona.example&"
```

Ten plik jest krótszą wersją tego ze strony Certyfikaty SSL: wystawienie, konwersja, weryfikacja: bez nazwy dostawcy kryptografii i z kluczem, którego nie da się wyeksportować (`Exportable = FALSE`), bo serwerowego klucza nie wynosimy poza serwer. Znaczenie pozostałych pól jest tam opisane.

SSL = Secure Sockets Layer: dawna nazwa szyfrowania, dziś formalnie TLS · **NT** = pierwotnie kryptonim N-Ten, od procesora, dla którego system miał powstać; popularnie dziś odczytywane jako New Technology, choć to rozwinięcie nie jest oficjalne: rdzeń dzisiejszego Windows Server · **CN** = Common Name: główna nazwa w certyfikacie, np. adres serwera · **OU** = Organizational Unit: folder w Active Directory grupujący konta i komputery · **IT** = Information Technology: dział albo praca związana z komputerami i infrastrukturą firmy · **RSA** = Rivest, Shamir, Adleman: algorytm szyfrowania kluczem publicznym, którym Vaultwarden podpisuje tokeny sesji, stąd pliki rsa_key* w danych aplikacji · **OID** = Object Identifier: numer, którym w SNMP wskazujemy jedną wartość urządzenia, na przykład stan baterii UPS

ZMIENIA SYSTEM

Wygenerowanie żądania z pliku odpowiedzi: certreq -new

```
certreq -new -machine request.inf vpn.csr
```

Przełącznik `-machine` musi pasować do `MachineKeySet = TRUE` w pliku odpowiedzi. Klucz prywatny powstaje teraz w magazynie komputera i nigdy nie opuszcza tego serwera.

certreq = narzędzie Windows do tworzenia i wysyłania żądań certyfikatów (CSR) do CA

ZMIENIA SYSTEM

Wysłanie żądania do CA-Zielona: certreq -submit -config

```
certreq -submit -config "dc01.ad.zielona.example\CA-Zielona" -attrib
"CertificateTemplate:ZielonaWWW" vpn.csr vpn.cer
```

(gdzie `dc01.ad.zielona.example\CA-Zielona` to nazwa serwera CA i nazwa urzędu, a `ZielonaWWW` to techniczna nazwa szablonu, nie nazwa wyświetlana; po przenosinach urzędu według strony Przenosiny urzędu z dc01 bez przerwy: dwa urzędy równolegle i wygaszenie CA-Zielona w tym i w kolejnych poleceniach wpisujemy `ca01.ad.zielona.example\Zielona-CA01`)

Przykład:

```
RequestId: 214
RequestId: "214"
Certificate retrieved(Issued) Issued
```

CA = Certificate Authority: urząd certyfikacji, który podpisuje certyfikaty · **certreq** = narzędzie Windows do tworzenia i wysyłania żądań certyfikatów (CSR) do CA

ZMIENIA SYSTEM

Wystawienie certyfikatu dla żądania z OpenSSL: certreq -submit

```
certreq -submit -config "dc01.ad.zielona.example\CA-Zielona" -attrib
"CertificateTemplate:ZielonaWWW" NazwaSerwera.csr NazwaSerwera.cer
```

Tak dostajemy certyfikat dla serwera z Linuksa: plik `.csr` z `docker01` kopiujemy na stację z Windowsem i wysyłamy do CA. Ponieważ klucz prywatny jest na `docker01`, nie robimy

`certreq -accept`, tylko kopiujemy `.cer` z powrotem na serwer. Szablon musi pozwalać na podanie nazwy w żądaniu (Supply in the request), inaczej CA próbuje zbudować nazwę z Active Directory, a serwera z Linuksa tam nie ma.

CA = Certificate Authority: urząd certyfikacji, który podpisuje certyfikaty · **certreq** = narzędzie Windows do tworzenia i wysyłania żądań certyfikatów (CSR) do CA

TYLKO ODCZYT

Odebranie certyfikatu po zatwierdzeniu: certreq -retrieve

```
certreq -retrieve -config "dc01.ad.zielona.example\CA-Zielona" NumerZadania vpn.cer
```

Używamy tego tylko wtedy, gdy `certreq -submit` odpowiedział, że żądanie czeka na decyzję (`Taken Under Submission`). Numer zadania (`RequestId`) pada w wyniku wysłania, a osoba z uprawnieniami menedżera CA zatwierdza żądanie w konsoli urzędu.

CA = Certificate Authority: urząd certyfikacji, który podpisuje certyfikaty · **certreq** = narzędzie Windows do tworzenia i wysyłania żądań certyfikatów (CSR) do CA

ZMIENIA SYSTEM

Zainstalowanie certyfikatu w magazynie komputera: certreq -accept

```
certreq -accept -machine vpn.cer
```

Certyfikat paruje się z kluczem prywatnym, który powstał przy `certreq -new`, i trafia do magazynu Osobiste komputera.

certreq = narzędzie Windows do tworzenia i wysyłania żądań certyfikatów (CSR) do CA

TYLKO ODCZYT

Sprawdzenie certyfikatu w magazynie: Get-ChildItem

```
Get-ChildItem Cert:\LocalMachine\My | Where-Object Subject -like "*vpn.zielona.example*"
| Format-List Subject, NotAfter, HasPrivateKey
```

Przykład:

```
Subject      : CN=vpn.zielona.example, OU=IT, O=Zielona sp. z o.o., L=Pamiatkowo,
S=Wielkopolskie, C=PL
NotAfter     : 3.10.2027 08:22:00
HasPrivateKey : True
```

Wartość `HasPrivateKey: False` oznacza, że certyfikat jest w magazynie, ale nie sparował się z kluczem, najczęściej dlatego, że `certreq -accept` poszedł na innym koncie albo innym komputerze niż `certreq -new`.

TYLKO ODCZYT

Lista szablonów wystawianych przez CA: certutil -CATemplates

```
certutil -config "dc01.ad.zielona.example\CA-Zielona" -CATemplates
```

Przykład:

```
ZielonaWWW: Zielona WWW (serwery) -- Auto-Enroll: Access is denied.
WebServer: Web Server -- Auto-Enroll: Access is denied.
Machine: Computer -- Auto-Enroll: Access is denied.
CertUtil: -CATemplates command completed successfully.
```

Komunikat `Auto-Enroll: Access is denied` dotyczy tylko automatycznego wnioskowania i nie przeszkadza w ręcznym wystawieniu, o ile nasze konto ma prawo Enroll do szablonu.

CA = Certificate Authority: urząd certyfikacji, który podpisuje certyfikaty · **certutil** = narzędzie Windows do sprawdzania, importowania i eksportowania certyfikatów z linii poleceń

TYLKO ODCZYT

Podgląd certyfikatu z CA: certutil -dump

```
certutil -dump vpn.cer
```

Wynik jest długi, więc niżej jest jego skrót z liniami, które sprawdzamy po wystawieniu.

Przykład (skrót):

```
Issuer: CN=CA-Zielona, DC=ad, DC=zielona, DC=example
 NotBefore: 3.10.2026 08:12
```

```
 NotAfter: 3.10.2027 08:22
Subject: CN=vpn.zielona.example, OU=IT, O=Zielona sp. z o.o., L=Pamiatkowo,
S=Wielkopolskie, C=PL
...
Subject Alternative Name
    DNS Name=vpn.zielona.example
```

□□ **CA** = Certificate Authority: urząd certyfikacji, który podpisuje certyfikaty · **certutil** = narzędzie Windows do sprawdzania, importowania i eksportowania certyfikatów z linii poleceń

□□ TYLKO ODCZYT

Weryfikacja łańcucha i unieważnień: certutil -verify -urlfetch

```
certutil -verify -urlfetch vpn.cer
```

Opcja `-urlfetch` każe sprawdzić łańcuch tak, jak zrobi to klient: pobiera certyfikaty urzędów z adresów zapisanych w certyfikacie (AIA) i listy unieważnień (CRL) z adresów CDP. Dzięki niej wychodzi na jaw to, czego nie widać w samym pliku: że CRL jest nieosiągalna albo wygasła.

Przykład (skrót):

```
Leaf certificate revocation check passed
CertUtil: -verify command completed successfully.
```

□□ **AIA** = Authority Information Access: miejsce w certyfikacie, które mówi, skąd pobrać certyfikat urzędu, który go podpisał · **CRL** = Certificate Revocation List: lista certyfikatów unieważnionych przed upływem ich terminu ważności · **CDP** = CRL Distribution Point: adres w certyfikacie, pod którym urząd publikuje listę unieważnionych certyfikatów · **certutil** = narzędzie Windows do sprawdzania, importowania i eksportowania certyfikatów z linii poleceń

□□ ZMIENIA SYSTEM

Certyfikat samopodpisany do laboratorium: openssl req -x509

```
openssl req -x509 -newkey rsa:3072 -nodes -keyout lab.key -out lab.crt -days 365 -config
san.cnf -extensions v3_req
```

Jednym poleceniem powstaje klucz i certyfikat podpisany tym samym kluczem. Używamy go tylko na maszynach laboratoryjnych, a każdy klient musi mieć `lab.crt` w swoim magazynie zaufanych, bo nikt poza nami go nie poświadcza.

□□ **openssl** = zestaw narzędzi do certyfikatów i szyfrowania

□□ ZMIENIA SYSTEM

Własna mała CA do testów: openssl req -x509 z CA:TRUE

```
openssl req -x509 -newkey rsa:4096 -nodes -keyout ca.key -out ca.crt -days 1825 -subj "/C=PL/O=Zielona sp. z o.o./CN=Zielona Lab CA" -addext "basicConstraints=critical,CA:TRUE" -addext "keyUsage=critical,keyCertSign,cRLSign"
```

Powstaje urząd laboratoryjny na pięć lat. Plik `ca.key` jest jego kluczem i trzymamy go poza serwerami, na których stoją certyfikaty, a nazwa `Zielona Lab CA` odróżnia go od prawdziwego CA-Zielona.

□□ **CA** = Certificate Authority: urząd certyfikacji, który podpisuje certyfikaty · **openssl** = zestaw narzędzi do certyfikatów i szyfrowania

□□ ZMIENIA SYSTEM

Podpisanie żądania własną CA: openssl x509 -req

```
openssl x509 -req -in NazwaSerwera.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out NazwaSerwera.crt -days 365 -sha256 -extfile san.cnf -extensions v3_req
```

Rozszerzenia (SAN, zastosowanie klucza) bierzemy z naszego `san.cnf`, a nie z żądania, bo to urząd decyduje, co trafia do certyfikatu. Alternatywą jest `-copy_extensions copy`, wygodniejsze, ale ufa temu, co wpisał wnioskodawca.

Przykład:

```
Certificate request self-signature ok
subject=C = PL, ST = Wielkopolskie, L = Pamiatkowo, O = Zielona sp. z o.o., OU = IT, CN = git.zielona.example
```

□□ **CA** = Certificate Authority: urząd certyfikacji, który podpisuje certyfikaty · **SAN** = w certyfikatach: Subject Alternative Name, dodatkowe nazwy hosta w certyfikacie; w pamięciach: Storage Area Network, sieć dyskowa · **openssl** = zestaw narzędzi do certyfikatów i szyfrowania

□□ ZMIENIA SYSTEM

Dodanie własnej CA do zaufanych na Rocky Linux: update-ca-trust

```
sudo cp ca.crt /etc/pki/ca-trust/source/anchors/zielona-lab-ca.crt
sudo update-ca-trust
```

Od tej chwili `curl`, `git` i inne programy na tym serwerze ufają certyfikatom podpisanym przez naszą laboratoryjną CA. Na stacjach domenowych tak samo rozdajemy certyfikat prawdziwego CA-Zielona, tylko przez zasady grupy, a nie ręcznie.

CA = Certificate Authority: urząd certyfikacji, który podpisuje certyfikaty · **cp** = copy: skopiuj · **update-ca-trust** = update CA trust: odświeżenie systemowego magazynu zaufanych urzędów certyfikacji w Rocky Linux

Z życia wzięte

Z ŻYCIA WZIĘTE

Środa, 14:10. Ola zgłasza, że `git clone` z serwera `docker01` kończy się błędem `SSL certificate problem: self-signed certificate`. Tydzień wcześniej Tomek na szybko wystawił dla `git.zielona.example` certyfikat samopodpisany, a w przeglądarkach wystarczyło kliknąć „Dalej", więc nikt się nie skarżył, dopóki nie wkroczyła konsola. Tomek generuje klucz i żądanie z `san.cnf`, wysyła je do CA-Zielona na `dc01`, a certyfikat wraca z szablonu `ZielonaWWW`. Stacje domenowe już ufają CA-Zielona, więc Ola nie musi niczego instalować.

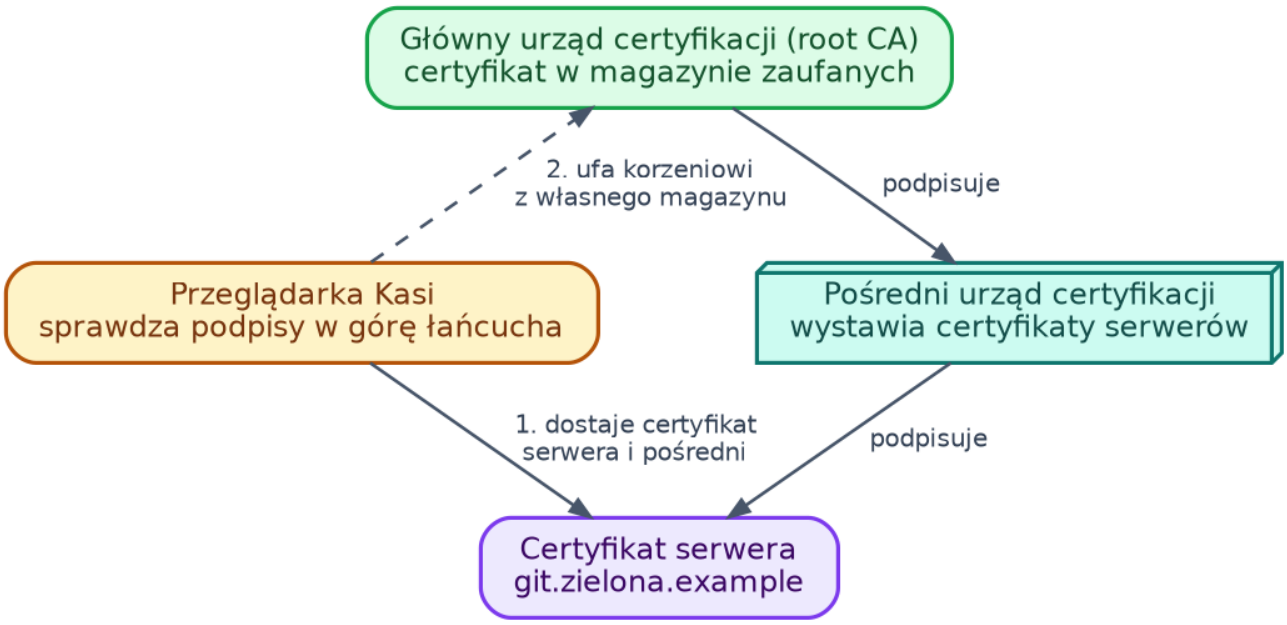
Jak to działa

Rozwiń: jak to działa pod spodem

Pojęcie: urząd certyfikacji (CA). Instytucja albo program, który podpisuje certyfikaty swoim kluczem prywatnym i tym samym poświadcza, że dany klucz publiczny należy do danego serwera. Klient ufa certyfikatowi tylko dlatego, że ufa temu, kto go podpisał, a nie dlatego, że sam certyfikat wygląda poprawnie.

Pojęcie: root CA i CA pośrednia. Root CA to certyfikat na samym szczycie zaufania, który leży w magazynie zaufanych na każdej stacji i w każdej przeglądarce. CA pośrednia dostaje podpis od roota i to ona na co dzień podpisuje certyfikaty serwerów, dzięki

czemu klucz roota może spokojnie leżeć wyłączony z sieci.



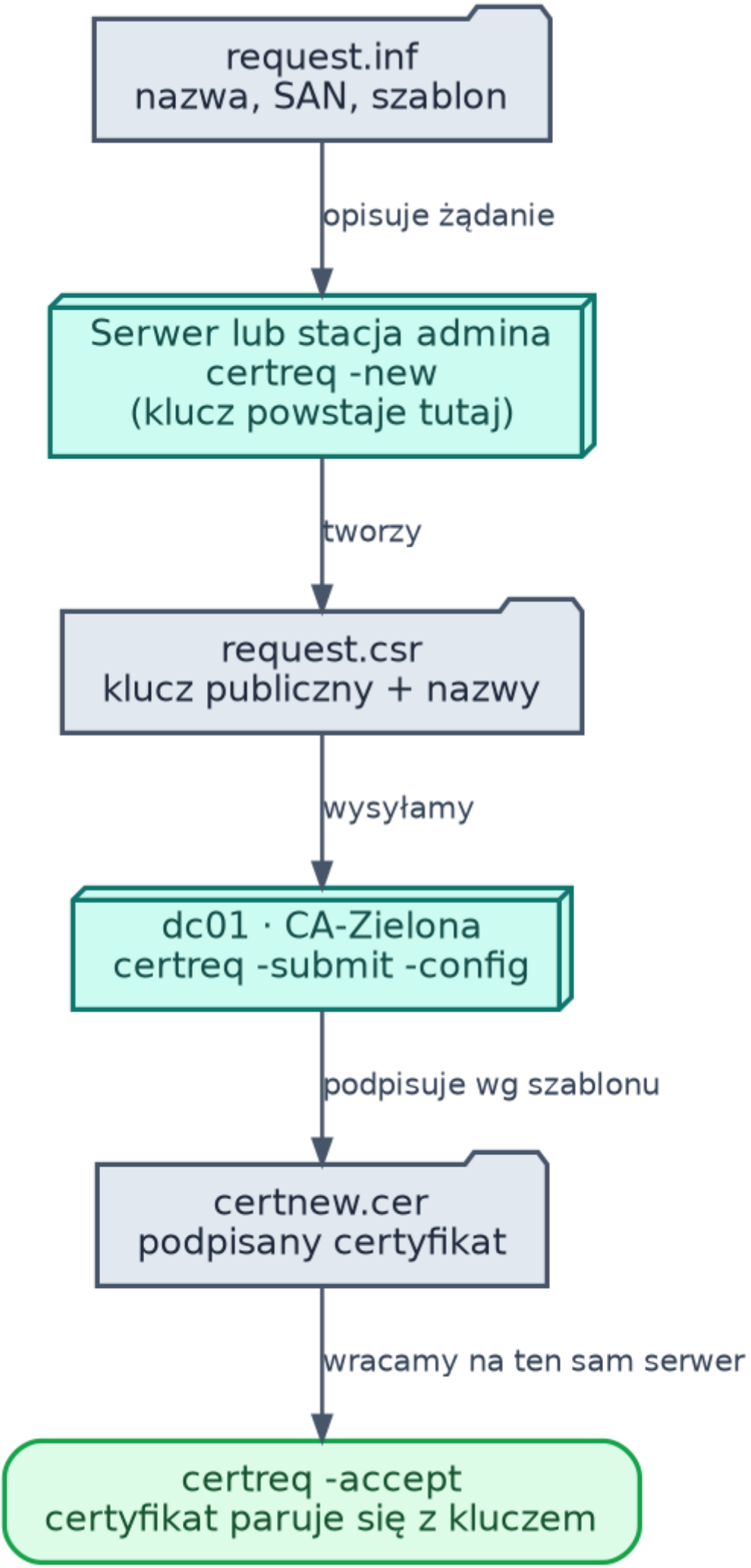
Drzewo zaufania: przeglądarka dostaje od serwera jego certyfikat i certyfikat pośredni, a korzeniowi ufa ze swojego magazynu (kliknięcie otwiera pełny rozmiar)

Przeglądarka Kasi nie pyta serwera, czy jest wiarygodny. Idzie od certyfikatu serwera w górę, sprawdza każdy podpis i zatrzymuje się dopiero na certyfikacie, który ma u siebie w magazynie zaufanych. Brak jednego ogniwa po drodze kończy się błędem, mimo że sam certyfikat serwera jest w porządku.

Pojęcie: AD CS. Active Directory Certificate Services, rola Windows Server, która zamienia serwer w urząd certyfikacji zintegrowany z domeną. W Zielonej pracuje na `dc01` jako urząd CA-Zielona, a stacje domenowe dostają jego certyfikat przez zasady grupy, więc ufają mu od razu. CA-Zielona ma jeden poziom: jest jednocześnie urzędem głównym i wydającym, więc jego klucz musi być cały czas w sieci. Docelową hierarchię z urzędem głównym offline i urzędem wydającym `ca01` opisuje strona Projekt PKI Zielonej: dlaczego nie na kontrolerze, hierarchia dwupoziomowa, nazwy, ważność, CRL i AIA.

Pojęcie: szablon certyfikatu. Gotowy przepis w AD CS, który mówi, co CA wolno wystawić: na jak długo, do czego ma służyć klucz, kto może o niego prosić i czy nazwę

bierzemy z Active Directory, czy z żądania. Szablon `ZielonaWWW` jest naszą kopią szablonu serwera WWW, z nazwą podawaną w żądaniu.


request.inf
nazwa, SAN, szablon
opisuje żądanie
Serwer lub stacja admina
certreq -new
(klucz powstaje tutaj)
tworzy
request.csr
klucz publiczny + nazwy
wysyłamy
dc01 · CA-Zielona
certreq -submit -config
podpisuje wg szablonu
certnew.cer
podpisany certyfikat
wracamy na ten sam serwer
certreq -accept
certyfikat paruje się z kluczem

Przepływ żądania w CA Windows: plik odpowiedzi, certreq -new, wysłanie do CA-Zielona, certnew.cer i certreq -accept (kliknięcie otwiera pełny rozmiar)

Kolejność jest zawsze ta sama: opisujemy żądanie, klucz powstaje na serwerze, do CA jedzie tylko żądanie, a z powrotem wraca podpisany certyfikat. Dla serwera z Linuksa pierwsze dwa kroki robi OpenSSL, a `certreq -accept` odpada.

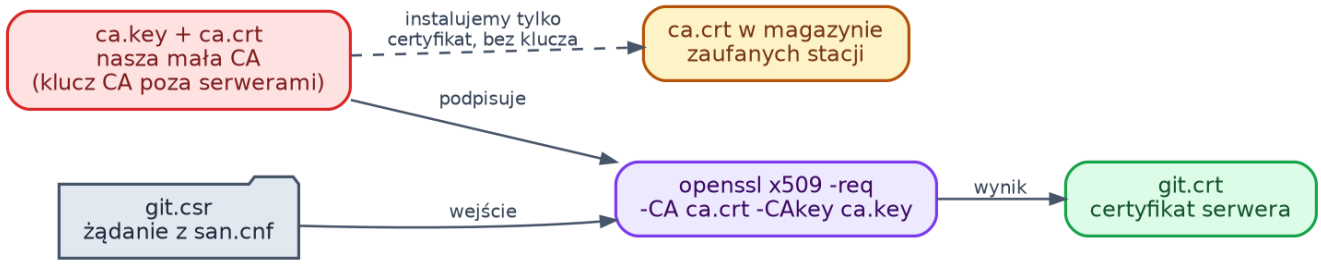
> ⚠ Na stronie Certyfikaty SSL: wystawienie, konwersja, weryfikacja pojawia się atrybut `san:dns=...` podawany przy wysyłaniu żądania. Działa on tylko wtedy, gdy na CA włączona jest flaga `EDITF_ATTRIBUTESUBJECTALTNAME2`, a tej flagi nie włączamy: pozwala podać dowolną nazwę w SAN każdemu, kto może wnioskować o certyfikat, i uchodzi za poważne ryzyko przejęcia kont. Zamiast niej używamy szablonu z nazwą podawaną w żądaniu, dostępnego wyłącznie dla administratorów.

Pojęcie: certyfikat samopodpisany. Certyfikat podpisany własnym kluczem, bez żadnej CA nad nim. Nikt poza właścicielem go nie poświadcza, więc każdy klient musi go dodać do zaufanych ręcznie, albo, co gorsze, klikać „Dalej" przy ostrzeżeniu.

W laboratorium jest w porządku, w produkcji nie, z trzech powodów, które widać w tabeli.

Problem	Skutek w produkcji
Brak CA nad certyfikatem	każdy klient wymaga ręcznego zaufania, a użytkownicy uczą się ignorować ostrzeżenia
Brak unieważnień	wycieku klucza nie da się ogłosić, certyfikat jest ważny do końca terminu
Brak porządku	nikt nie wie, ile certyfikatów mamy i kiedy wygasają

Własna mała CA z OpenSSL jest o jeden krok wyżej: klienci ufają jednemu plikiem `ca.crt`, a nie każdemu certyfikatowi osobno. Nadal nie ma tu unieważnień ani audytu, więc to narzędzie do prób, nie do firmy.



Mała CA z OpenSSL: klucz i certyfikat CA podpisują żądanie serwera, a stacje dostają tylko ca.crt (kliknięcie otwiera pełny rozmiar)

Częste błędy

Objaw	Przyczyna	Co robimy
Certyfikat z CA-Zielona ma tylko CN i brak SAN	szablon buduje nazwę z Active Directory zamiast z żądania	wybieramy szablon z nazwą podawaną w żądaniu, sprawdzamy wynik `certutil -dump`
`certreq -submit` kończy się `Taken Under Submission`	szablon wymaga zatwierdzenia przez menedżera CA	zatwierdzamy żądanie w konsoli urzędu, potem `certreq -retrieve NumerZadania`
Błąd `0x80094800` przy wysyłaniu żądania	CA nie wystawia tego szablonu albo nasze konto nie ma prawa Enroll	`certutil -config ... -CATemplates`, w konsoli szablonów sprawdzamy publikację i uprawnienia
`unable to get local issuer certificate` w `openssl verify`	brakuje certyfikatu pośredniego	dokładamy go przez `-untrusted` albo do pliku `fullchain.pem`
`certutil -verify -urlfetch` zgłasza `0x80092013` (serwer unieważnień offline)	adres CDP z certyfikatu jest nieosiągalny albo CRL wygasła	publikujemy nową CRL, adres sprawdzamy w `certutil -dump`
Po `certreq -accept` brak klucza (`HasPrivateKey: False`)	`-new` i `-accept` wykonane na różnych kontach albo komputerach, albo bez `-machine`	powtarzamy oba kroki na tym samym komputerze i w tym samym kontekście
Szablon odrzuca klucz EC	szablon dopuszcza tylko RSA	prosimy o klucz RSA albo przygotowujemy szablon z ECDSA
`openssl x509 -req` daje certyfikat bez SAN	brak `-extfile` i `-extensions` albo `-copy_extensions copy`	dopisujemy `-extfile san.cnf -extensions v3_req`

Ćwiczenia

1. Wygeneruj klucz RSA i żądanie z pliku `san.cnf` dla nazwy `test.zielona.example`, a potem sprawdź, jakie nazwy trafiły do żądania.

Odpowiedź

W `san.cnf` zmieniamy `CN` i obie pozycje w `[alt_names]`, potem wykonujemy `openssl req -new -newkey rsa:3072 -nodes -keyout test.key -out test.csr -config san.cnf`. Nazwy sprawdzamy poleceniem `openssl req -in test.csr -noout -text | grep DNS:`.

1. Zbuduj małą CA, podpisz nią żądanie z ćwiczenia 1 i zweryfikuj wynik. Co zobaczymy, jeśli w `openssl verify` pominiemy `-CAfile`?

Odpowiedź

Wykonujemy kolejno `openssl req -x509 ...` (CA), `openssl x509 -req ... -CA ca.crt -CAkey ca.key ...` (podpis) i `openssl verify -CAfile ca.crt test.crt`, co daje `test.crt: OK`. Bez `-CAfile` OpenSSL szuka korzenia w systemowym magazynie zaufanych, nie znajduje naszej laboratoryjnej CA i kończy błędem `unable to get local issuer certificate`.

1. Dlaczego przy certyfikacie dla serwera z Linuksa nie robimy `certreq -accept`?

Odpowiedź

`certreq -accept` paruje certyfikat z kluczem prywatnym zapisanym w magazynie Windows. Klucz serwera z Linuksa nigdy nie był w Windowsie, bo powstał przez OpenSSL na `docker01`, więc nie ma z czym parować. Gotowy plik `.cer` po prostu kopiujemy na serwer, gdzie leży klucz.

1. Po wdrożeniu certyfikatu na `vpn.zielona.example` Kasia dalej widzi ostrzeżenie w przeglądarce. Jakie dwa polecenia uruchamiamy najpierw, żeby ustalić, czy winien jest serwer, czy łańcuch?

Odpowiedź

Najpierw `openssl s_client -connect vpn.zielona.example:443 -servername vpn.zielona.example -CAfile CA-Zielona.crt -verify_return_error`, żeby zobaczyć, co faktycznie podaje serwer i czy przechodzi weryfikację. Potem `certutil -verify -urlfetch` na pliku certyfikatu, żeby sprawdzić, czy klient dotrze do CRL i certyfikatów urzędów z adresów zapisanych w certyfikacie.

Powiązane: Certyfikaty SSL: wystawienie, konwersja, weryfikacja · Apache Solr w Dockerze: instalacja, rdzeń i pierwsze wyszukiwanie · Serwer SFTP · MailCow: konta z Active Directory

Źródła: docs.openssl.org, strony `openssl-req`, `openssl-x509`, `openssl-verify`, `openssl-s_client` · learn.microsoft.com, `certreq` i `certutil` (Windows Commands) · learn.microsoft.com, Active Directory Certificate Services

---

Revision #17
Created 2026-10-03 09:39:04 UTC by Tomasz Konieczny
Updated 2026-10-26 14:14:32 UTC by Tomasz Konieczny