

Podpisanie i weryfikacja kodu aplikacji w systemie Windows

Martyna Borkowska, Karolina Glaza, Amila Amarasekara
<https://github.com/kequel>

Maj 2025

Zagadnienia

1. Demonstracja certyfikatu wystawionego przez dowolne CA z właściwymi rozszerzeniami (v3 Extensions) oraz zaufania do CA, który go wystawił na testowej maszynie.
2. Demonstracja podpisywania kodu w dowolnym IDE (przykładowo VisualStudio, Eclipse) - demonstracja weryfikacji podpisu.
3. Demonstracja podpisywania kodu ręcznie w oparciu o dowolne narzędzie.
4. Demonstracja ręcznego podpisywania kodu z użyciem publicznie dostępnej usługi znakowania czasem (podpisu odpornego na nieaktualność certyfikatu).

Wstęp teoretyczny

Podpis cyfrowy aplikacji

sposób zapewnienia, że:

- kod pochodzi od zaufanego wydawcy (autentyczność)
- nie został zmodyfikowany po podpisaniu (integralność)
- może być zaufany przez system operacyjny

Systemy operacyjne coraz częściej wymagają, by aplikacje były podpisane - szczególnie przy dystrybucji przez internet. Nieużycie podpisu może skutkować:

- ostrzeżeniami bezpieczeństwa ("Nieznany wydawca")
- blokadą instalacji przez system

Działanie

1. Wydawca aplikacji generuje parę kluczy: prywatny i publiczny.
2. Certyfikat X.509 (np. od CA jak DigiCert, Sectigo) zawiera klucz publiczny i dane właściciela.
3. Kod jest podpisywany kluczem prywatnym - hash + podpis.
4. System używa klucza publicznego z certyfikatu do weryfikacji podpisu.

Timestamp (znacznik czasu)

To dowód, że podpis był ważny w momencie jego złożenia, nawet jeśli certyfikat wygasł później.

Wymaga połączenia z usługą TSA (Time Stamping Authority), np.:

- <http://timestamp.digicert.com>
- <http://timestamp.sectigo.com>

1 Generowanie własnego Urzędu Certyfikacji (CA)

Generowanie własnego Urzędu Certyfikacji (CA) to proces tworzenia zaufanego podmiotu, który będzie wystawiał certyfikaty dla aplikacji.

1.1 Wymagane narzędzia:

- **OpenSSL** działające w terminalu – narzędzie kryptograficzne,
- **PowerShell z uprawnieniami administratora** - do wykonania komend systemowych,
- **Plik konfiguracyjny** (opcjonalnie) – dla zaawansowanych ustawień CA.

1.2 Procedura generowania CA

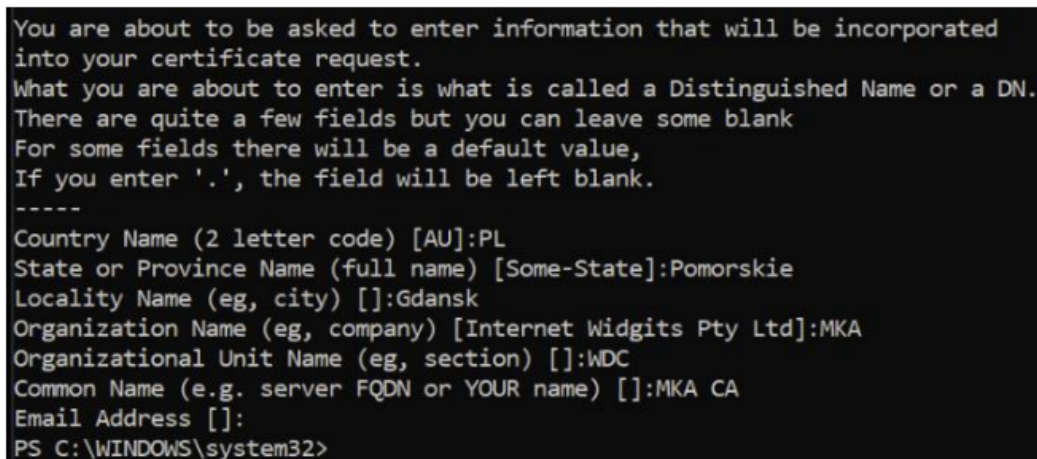
1. Uruchom PowerShell jako administrator.
2. Wykonaj komendę generującą klucz i certyfikat CA:

```
openssl req -x509 -newkey rsa:2048 \  
-keyout ca.key -out ca.crt \  
-days 365 \  
-addext "basicConstraints=CA:TRUE"
```

Listing 1: Generowanie CA w PowerShell

Parametry:

- `x509` - generuje certyfikat samopodpisany (X.509),
- `newkey rsa:2048` - tworzy klucz RSA 2048-bitowy,
- `days 365` - okres ważności certyfikatu,
- `addext` - dodaje rozszerzenia krytyczne dla CA



```
You are about to be asked to enter information that will be incorporated  
into your certificate request.  
What you are about to enter is what is called a Distinguished Name or a DN.  
There are quite a few fields but you can leave some blank  
For some fields there will be a default value,  
If you enter '.', the field will be left blank.  
-----  
Country Name (2 letter code) [AU]:PL  
State or Province Name (full name) [Some-State]:Pomorskie  
Locality Name (eg, city) []:Gdansk  
Organization Name (eg, company) [Internet Widgits Pty Ltd]:MKA  
Organizational Unit Name (eg, section) []:WDC  
Common Name (e.g. server FQDN or YOUR name) []:MKA CA  
Email Address []:  
PS C:\WINDOWS\system32>
```

Rysunek 1: Uzupełnione dane.

1.3 Wygenerowane pliki

```
PS C:\WINDOWS\system32> dir ca.*

Directory: C:\WINDOWS\system32

Mode                LastWriteTime         Length Name
----                -
-a----             19.05.2025     22:28         1336 ca.crt
-a----             19.05.2025     22:23         1916 ca.key
```

Rysunek 2: Struktura plików CA po generacji

Po wykonaniu komendy powstają:

- **ca.crt** - certyfikat publiczny CA:
 - Zawiera klucz publiczny i metadane,
 - Może być dystrybuowany do zaufanych magazynów.
- **ca.key** - klucz prywatny CA. Jeżeli zaczyna się od:
 - **-----BEGIN ENCRYPTED PRIVATE KEY-----** - klucz jest zaszyfrowany
 - **-----BEGIN PRIVATE KEY-----** - nie ma ochrony hasłem

```
ca.key
1 -----BEGIN ENCRYPTED PRIVATE KEY-----
2 MIIFNTBFBgkqhkiG9w0BBQ0wUjAxBgkqhkiG9w0BBQwwJAQQkTitLv0MTf7Jh8Bt
3 W8GgnQICCAAwDAYIKoZIhvcNAgkFADAdBg1ghkgBZQMEASoEEOb8BJV5g2b/o2b9
4 9LOAMXcEggTQXo0GgaGSL4r8V9Pr1qOlyMI0wQVfiXxF1gkYvhUctKrxDaOZKLAU
5 bXvx72a+TtRe0Hd8Cgc0EcK1qAPQPuImtonAB0MdDwW8me+8FmOuxQaTySx0Oow
6 TtuD/ZII6sVqZBNWtNeV/E30P98xpK+qTpxZhoZ0jd4J+0ciBdjUn/AzyxVm6dqw
7 g5iJNEuS2JRCASIV8bc1oJ5nAwRk2BN9pR+RiLa4Vj2z0zGtCALvRT/aVv4vfh5h
8 TmOsLFs9ZYizgnyjpJz27tR0zjiHZ6EH3cnMcWZGJKvJUYxpJ0WmymWkWFrcfg
9 OU2I9r76VO9TnSd521h/6U38hb6ZnewKcQo22Bx2QxQEL84eJy4GjFWIE4IwrV5e
10 +3efE6Rv2K5Lbp9wTA2DiDfjX3+IePlVBEILmkjALaYrQYhRuUJbvDjo+T4skWUY
11 aMfctnVvHieOUuibyoZ200uleK0ErBy3ZJsaPTLDHIQqWaRebJLQ1EC38NoQZzfc
12 CauV2PCwJqx6Dzy4cgS4SYDHPs2VXIXTW1+ZiY9pB0DdpElfR5C3BAkqy3Bop0Ku
13 ko1Av7xZYJuXLC57yizi5qayYMG5lqyyB20cJFucx14kuK62ryf6WtdpxvQFA62d
14 JLRJzcdRqbYL+MNFxonnBJ5g9G1jwkoWtxkNr1sEN3eN3A1UbT3Z+XfTSdrtQt4y
15 d2FACAFGRs8BDytOFrht7a9u7Hv2aVrVYamTgNsgDec7FigtZkrihJx/WQ4sf4Qx
16 gtxjntgovZNZgyiackyugvD+WkRhNx8E87YhC5kTXjeqf04czHBuZMI+fV0VKF3
17 I022YGCKFhpy2FKcIy/711x4zd1FcXPT+0kKRXDeRnICSJQxwjjoznISu6rIk0k5C
18 R8Afib3Zkk6uS21/s+OKP5L6on1mqQEzgvnshktMDAr27YhNu0Y11n7WB2yNb9hH
19 V6pCgbw2s0SoiJ3bLMY4Z4nV/G+kJL6aeXdaUqQ+7MUUnt5iTL4CXbrHE/N87wa
20 11s6k/TBB11fNJzNKu5/srP+t8tvnbWcabmqwYrp2/aw5oINMXTHwisdmjwQSWnN
21 fnSIE40QGHbMeBCp03GPOUvet2tqraaA9DB1fz6D2bajRw1kAiUi66gT0ShP7ncB
22 IHJNX2cDdT6r6xxCr7K4/4+aDW8sRnupm9RhJ/qw5id6DqluYwkiFno/a9SbTt
23 eRRQ8eZjS0rI7Q4QGw19eUak14NyQ8nQ7HfK/NER6jNTa73hk/Xkrk32/cjGeDzr
24 7tBfyao2v5wyvf4CXDpz2UdbVvg4Sd2LJhYpExVCs+/ToTc2i+IYMB0L5MmySWH/
25 sw3sW0RoZ1Y/SqtsM9jKB5aLoUknmh4Y+ph6v8D+kMyG2rgmZKlYs9PK92rCgSoA
26 qLShjoeKM9PVD7GAlIRK6NVlTHXhL3nOTbeRfUz8CGrfgyBZKJpWwF4eU4MqKZQB
27 q5MP9azYpHQ69Ylm3Xbpo1gDU/vq40eVRjmU1KoI4nnEX+XLzIysGGZjrm0dBS8E
28 lYx58V+A2R5oDxWLLSzoCFFzxkyQ2BNAqzcEua8cuNAwa9aQ95jitsATZFniZCZ6
29 RWJyo4OnHyGpLRmQp9siZ6LC+wNp1DmGdFgTuqevqftrRQLdMrZUw7E=
30 -----END ENCRYPTED PRIVATE KEY-----
31
```

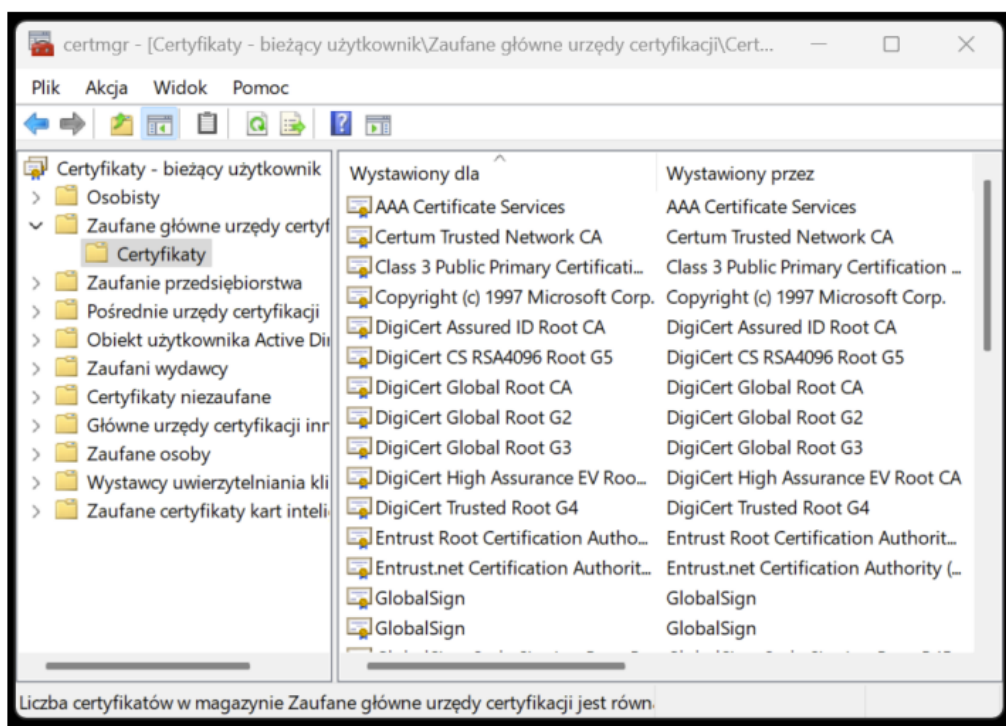
Rysunek 3: Plik ca.key.

2 Dodanie CA do zaufanych certyfikatów w Windows

Aby system Windows ufał certyfikatom podpisanym przez nasze CA, konieczne jest dodanie certyfikatu `ca.crt` do magazynu **Zaufane główne urzędy certyfikacji**. W tym celu należy uruchomić `certmgr.msc` – narzędzie MMC do zarządzania certyfikatami użytkownika, z uprawnieniami administratora.

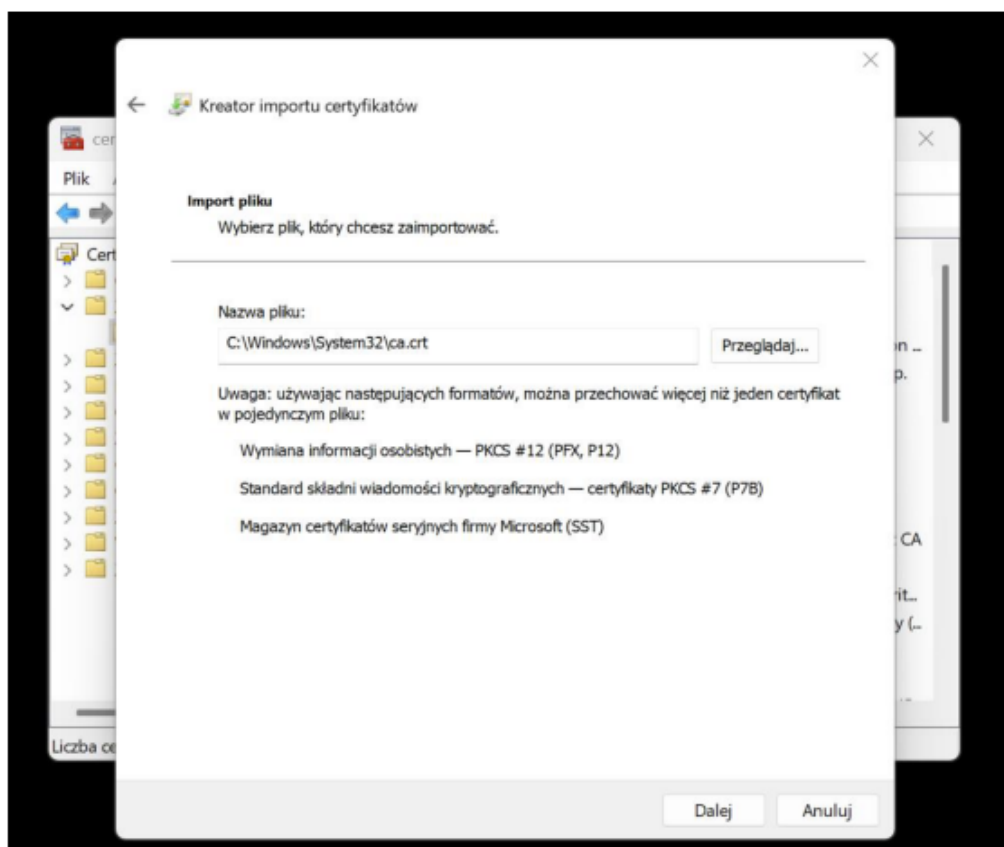
2.1 Procedura importu CA

1. Otwórz `certmgr.msc` poprzez:
 - Menu Start → "Zarządzanie certyfikatami", lub
 - Uruchom → `certmgr.msc`
2. W drzewie nawigacyjnym wybierz:
Certyfikaty → Bieżący użytkownik
→ Zaufane główne urzędy certyfikacji
→ Certyfikaty



Rysunek 4: Interfejs certmgr.msc.

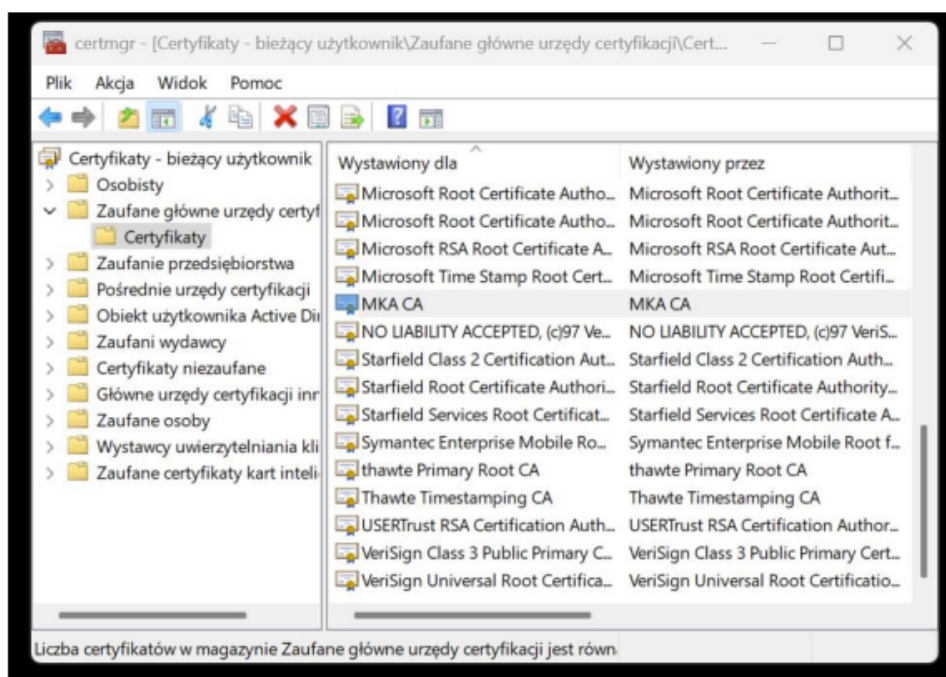
3. Kliknij prawym przyciskiem i wybierz **Wszystkie zadania** → **Importuj...**
4. W kreatorze importu:
 - Wskaż plik `ca.crt`,
 - Wybierz magazyn **Zaufane główne urzędy certyfikacji**,
 - Potwierdź operację.



Rysunek 5: Wybór ca.crt w certmgr.msc.

2.2 Weryfikacja poprawności importu

W narzędziu sprawdź, czy certyfikat pojawił się na liście zaufanych CA:



Rysunek 6: Lista zaufanych certyfikatów z widocznym certyfikatem "MKA CA".

2.3 Konsekwencje

- System będzie ufał wszystkim certyfikatom podpisanym przez ten CA,
- Klucz `ca.key` musi być chroniony - jego kompromitacja oznacza utratę zaufania do całej infrastruktury,

3 Generowanie certyfikatu dla aplikacji

Aby możliwe było podpisanie aplikacji, należy wygenerować parę kluczy oraz certyfikat podpisany przez wcześniej utworzony CA. Proces ten składa się z trzech głównych kroków: wygenerowania żądania certyfikatu (CSR), jego podpisania oraz konfiguracji rozszerzeń X.509.

3.1 Generowanie klucza i żądania CSR

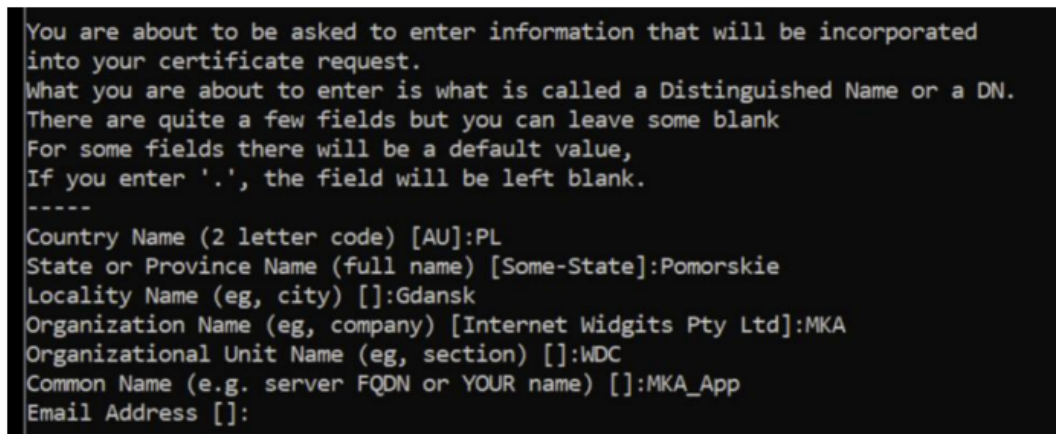
W pierwszym kroku generujemy klucz prywatny aplikacji oraz żądanie certyfikatu (CSR):

```
openssl req -newkey rsa:2048 -keyout app.key -out app.csr -nodes
```

Listing 2: Generowanie klucza i CSR

Opis parametrów:

- `newkey rsa:2048` - generuje nowy klucz RSA o długości 2048 bitów,
- `keyout app.key` - zapisuje klucz prywatny do pliku,
- `out app.csr` - tworzy żądanie certyfikatu (CSR),
- `nodes` - nie szyfruje klucza prywatnego (brak hasła).



```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:PL
State or Province Name (full name) [Some-State]:Pomorskie
Locality Name (eg, city) []:Gdansk
Organization Name (eg, company) [Internet Widgits Pty Ltd]:MKA
Organizational Unit Name (eg, section) []:WDC
Common Name (e.g. server FQDN or YOUR name) []:MKA_App
Email Address []:
```

Rysunek 7: Wprowadzenie danych do CSR

W wyniku wykonania tej komendy powstają dwa pliki:

- `app.key` - zawiera klucz prywatny aplikacji. Jest to klucz, którym będzie podpisywany kod. Należy chronić go przed nieautoryzowanym dostępem, ponieważ jego utrata lub wyciek może zagrozić bezpieczeństwu całego procesu podpisywania.
- `app.csr` - żądanie podpisania certyfikatu (Certificate Signing Request). Zawiera dane identyfikacyjne (np. nazwa organizacji, CN, kraj) oraz klucz publiczny odpowiadający prywatnemu z `app.key`. Plik ten jest przeznaczony do przekazania urzędowi certyfikacji (CA) w celu wystawienia certyfikatu.


```
PS C:\WINDOWS\system32> dir app.*

Directory: C:\WINDOWS\system32

Mode                LastWriteTime         Length Name
----                -
-a----             19.05.2025      23:04         1076 app.csr
-a----             19.05.2025      23:01         1732 app.key
```

Rysunek 8: Wygenerowane pliki: app.key i app.csr

3.2 Podpisanie CSR przez CA

Następnie podpisujemy CSR przy użyciu wcześniej wygenerowanego urzędu CA:

```
openssl x509 -req -in app.csr \
  -CA ca.crt -CAkey ca.key \
  -CAcreateserial \
  -out app.crt \
  -days 365 \
  -extfile v3.ext
```

Listing 3: Podpisywanie CSR przez CA

Opis parametrów:

- req -in app.csr - określa CSR do podpisania,
- CA ca.crt -CAkey ca.key - pliki CA używane do podpisu,
- CAcreateserial - tworzy plik z numerem seryjnym,
- out app.crt - wynikowy certyfikat aplikacji,
- days 365 - ważność certyfikatu (w dniach),
- extfile v3.ext - plik z rozszerzeniami certyfikatu.

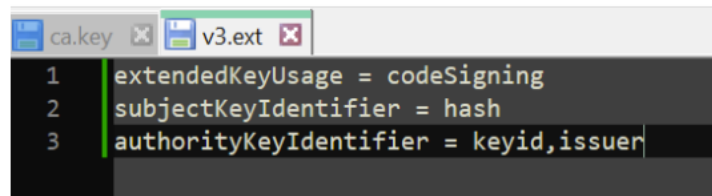
```
PS C:\WINDOWS\system32> openssl x509 -req -in app.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out a
pp.crt -days 365 -extfile v3.ext
Certificate request self-signature ok
subject=C=PL, ST=Pomorskie, L=Gdansk, O=MKA, OU=WDC, CN=MKA_App
Enter pass phrase for ca.key:
```

Rysunek 9: Proces podpisywania certyfikatu

3.3 Plik konfiguracyjny v3.ext

Podczas podpisywania żądania CSR konieczne jest podanie pliku z rozszerzeniami X.509v3, które okre-
ślają dodatkowe właściwości certyfikatu. Taki plik nazywa się najczęściej v3.ext i zawiera zestaw dy-
rektyw opisujących, jak systemy powinny interpretować dany certyfikat.

Przykładowa zawartość pliku v3.ext:



```
ca.key v3.ext
1 extendedKeyUsage = codeSigning
2 subjectKeyIdentifier = hash
3 authorityKeyIdentifier = keyid,issuer
```

Rysunek 10: Zawartość pliku v3.ext

Opis kluczowych rozszerzeń:

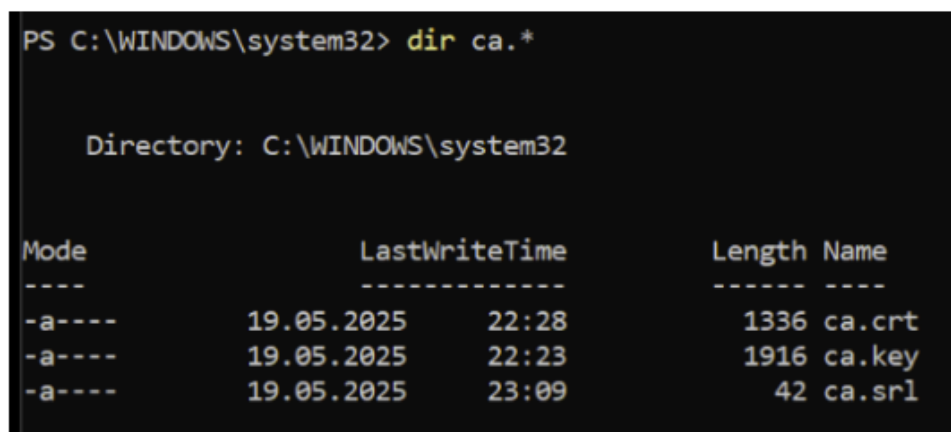
- `extendedKeyUsage = codeSigning` - informuje, że certyfikat może być używany do podpisywania kodu źródłowego lub wykonywalnego. Jest to wymagane przez narzędzia takie jak `signtool`.
- `subjectKeyIdentifier = hash` - dodaje identyfikator klucza publicznego w postaci skrótu, co ułatwia zarządzanie i sprawdzanie powiązań w łańcuchu certyfikatów.
- `authorityKeyIdentifier = keyid,issuer` - wskazuje na certyfikat nadrzędny (czyli CA), który wydał ten certyfikat. Umożliwia prawidłową budowę i walidację łańcucha zaufania.

Tego typu plik jest standardem w procesie tworzenia certyfikatów X.509 i zapewnia, że certyfikat zostanie prawidłowo rozpoznany przez oprogramowanie systemowe, przeglądarki, IDE i inne narzędzia korzystające z infrastruktury klucza publicznego.

3.4 Wynik operacji

Po wykonaniu wszystkich kroków, otrzymujemy zestaw plików gotowych do użycia w procesie podpisywania aplikacji:

- `app.key` - klucz prywatny aplikacji,
- `app.crt` certyfikat podpisany przez CA,
- `ca.srl` - numer seryjny podpisanego certyfikatu.



```
PS C:\WINDOWS\system32> dir ca.*

Directory: C:\WINDOWS\system32

Mode                LastWriteTime         Length Name
----                -
-a----          19.05.2025   22:28           1336 ca.crt
-a----          19.05.2025   22:23           1916 ca.key
-a----          19.05.2025   23:09             42 ca.srl
```

Rysunek 11: Ostateczne pliki: certyfikat aplikacji i pliki CA

4 Podpisywanie kodu w Visual Studio

4.1 Przygotowanie certyfikatu PFX

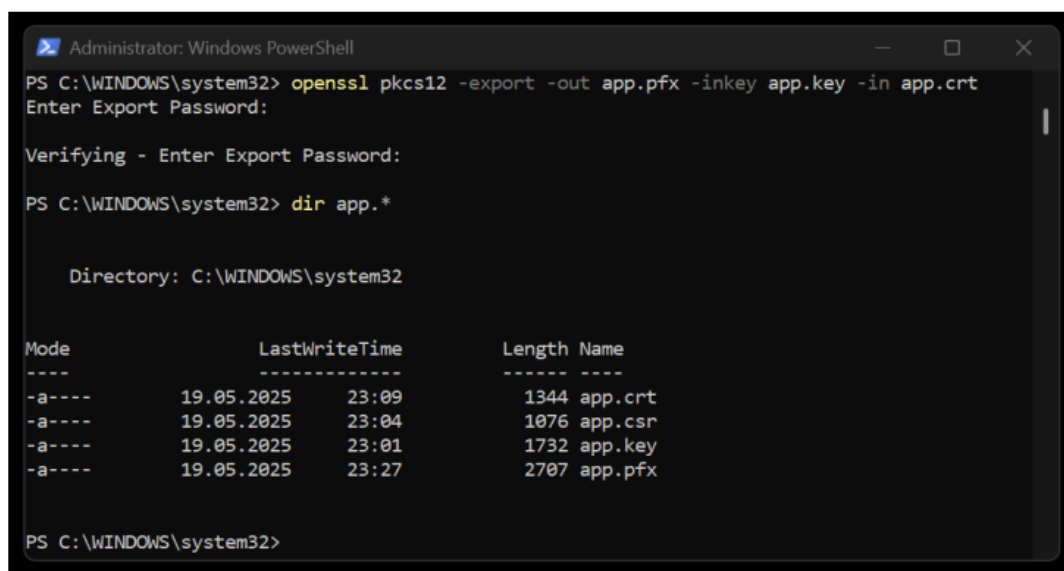
Plik PFX (.pfx) jest niezbędny do podpisywania aplikacji w środowiskach takich jak Visual Studio. Format ten łączy w jednym pliku certyfikat (.crt) oraz odpowiadający mu klucz prywatny (.key), opcjonalnie chroniąc je hasłem. Do jego utworzenia służy polecenie:

```
openssl pkcs12 -export -out app.pfx -inkey app.key -in app.crt
```

Listing 4: Generowanie PFX

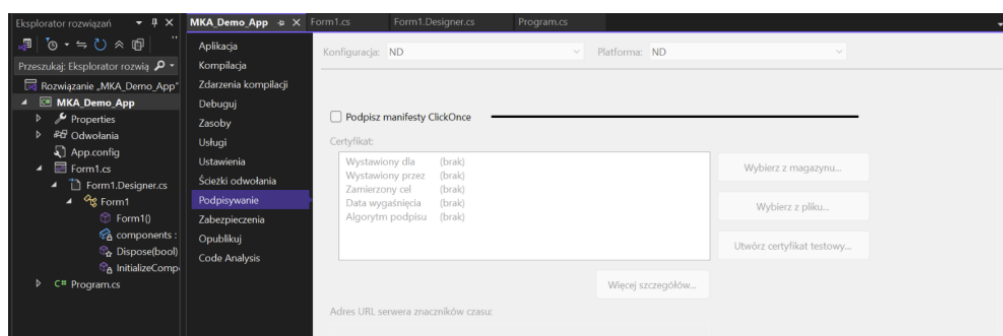
Opis parametrów:

- export - tryb eksportu: tworzy nowy plik PFX,
- out app.pfx - nazwa wynikowego pliku PFX,
- inkey app.key - wskazuje klucz prywatny aplikacji,
- in app.crt - wskazuje certyfikat, który zostanie dołączony.

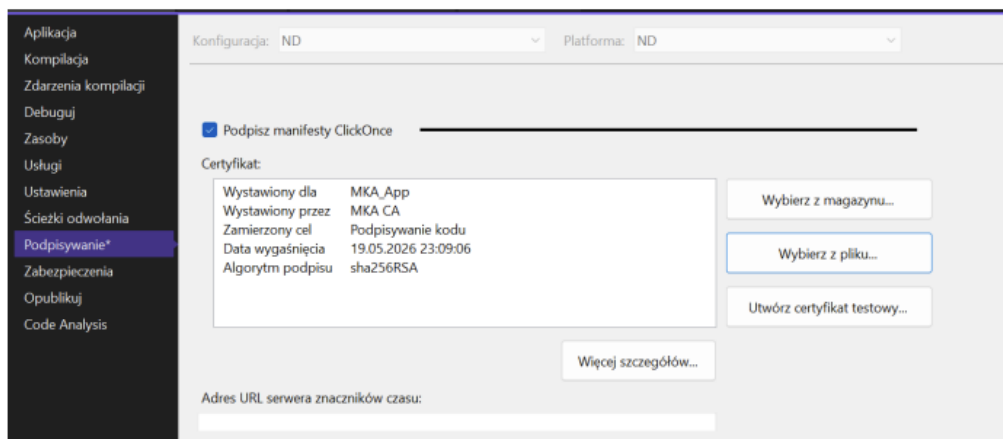


Rysunek 12: Generowanie pliku PFX w PowerShell oraz wygenerowany plik.

4.2 Próba podpisania w Visual Studio



Rysunek 13: Konfiguracja podpisywania w zakładce właściwości.



Rysunek 14: Po podpisaniu plikiem app.pfx.

Mimo, że usług Visual Studio podpisany, w właściwościach aplikacji w bin/debug podpis nie będzie widoczny. Eksplorator plików pokazuje podpisy tylko od zaufanych urzędów certyfikacji (np. DigiCert). Certyfikat self-signed może być uznany za "nieznany", należy więc przeprowadzić ręczne podpisywanie.

5 Ręczne podpisywanie kodu z timestampem

5.1 Proces podpisywania signtool

Do ręcznego podpisywania aplikacji w systemie Windows wykorzystujemy narzędzie `signtool.exe`, które jest częścią zestawu Windows SDK. Poniższa komenda podpisuje plik wykonywalny przy użyciu wcześniej wygenerowanego certyfikatu w formacie PFX oraz dodaje znacznik czasu z serwera TSA.

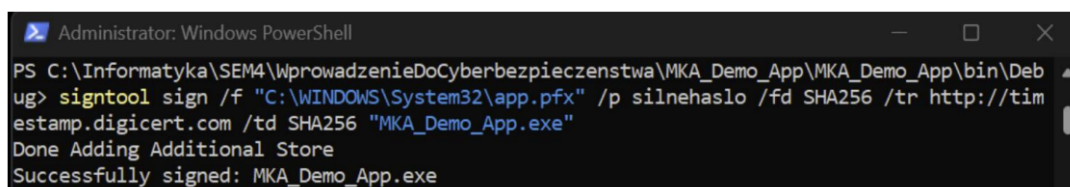
```
signtool sign /f "C:\WINDOWS\System32\app.pfx" /p silnehaslo
/fd SHA256 /tr http://timestamp.digicert.com /td SHA256
"MKA_Demo_App.exe"
```

Listing 5: Podpisywanie signtool

Opis parametrów:

- `sign` - polecenie podpisania pliku,
- `/f` - ścieżka do pliku PFX zawierającego certyfikat i klucz prywatny,
- `/p` - hasło do pliku PFX,
- `/fd SHA256` – algorytm skrótu (hashowania) używany do tworzenia podpisu,
- `/tr http://timestamp.digicert.com` - adres serwera TSA (Time Stamping Authority), który dostarcza znacznik czasu,
- `/td SHA256` - algorytm skrótu używany przy żądaniu znacznika czasu,
- `MKA_Demo_App.exe` - podpisywany plik wykonywalny.

Zastosowanie znacznika czasu (`/tr`) jest kluczowe - dzięki niemu podpis zachowuje ważność nawet po wygaśnięciu certyfikatu. Algorytm SHA256 zapewnia nowoczesny poziom bezpieczeństwa i jest zgodny z aktualnymi wymaganiami Microsoftu.



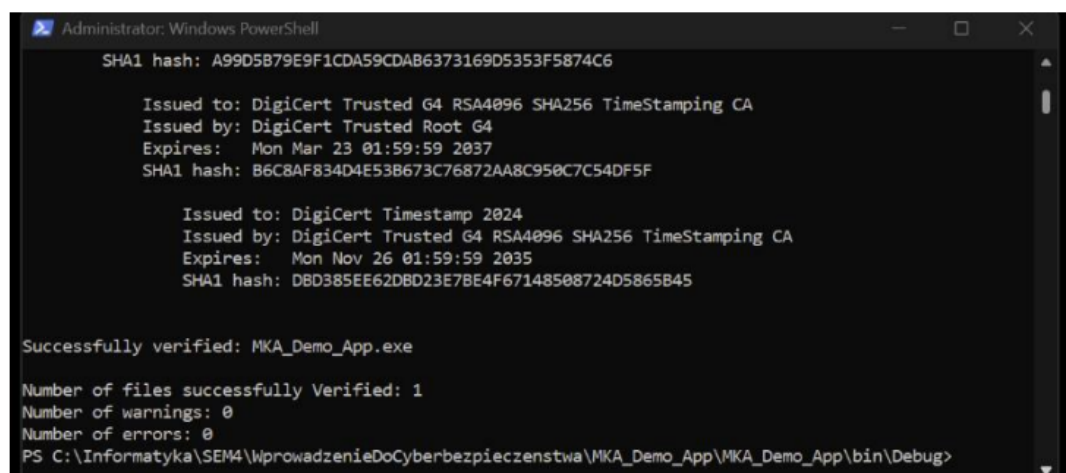
Rysunek 15: Ręczne podpisywanie aplikacji.

5.2 Weryfikacja podpisu

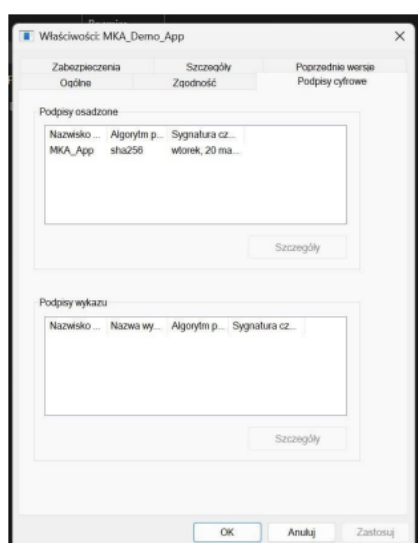
Sprawdzono poprawność podpisu poprzez właściwości pliku w Eksploratorze Windows (zakładka „Podpisy cyfrowe”) oraz za pomocą:

```
signtool verify /v /pa "MKA_Demo_App.exe"
```

Listing 6: Weryfikacja podpisu



Rysunek 16: Weryfikacja podpisu w PowerShell.



Rysunek 17: Weryfikacja podpisu cyfrowego w oknie właściwości pliku.

5.3 Techniczne aspekty timestampingu

- Serwer TSA: `http://timestamp.digicert.com`
- Algorytm hashowania: SHA256
- Weryfikacja łańcucha certyfikatów TSA

6 Skrypt automatyzujący proces

W ramach projektu zdecydowaliśmy się napisać także krótki skrypt automatyzujący cały powyższy proces podpisywania kodu, realizując wszystkie kluczowe kroki opisane w projekcie:

- Generację własnego Urzędu Certyfikacji (CA)
- Dodawanie CA do zaufanych certyfikatów w Windows
- Tworzenie certyfikatu dla aplikacji
- Generację pliku PFX
- Podpisywanie plików wykonywalnych z timestampem
- Weryfikację poprawności podpisu

6.1 Parametry skryptu (wymagają uzupełnienia odpowiednimi danymi):

```
$caName = "MKA2 CA"           # Nazwa urzedu certyfikacji
$appName = "MKA_App2"         # Nazwa aplikacji
$password = "silnehaslo"      # Haslo do certyfikatow
$exePath = "sciezka/do/aplikacji" # Sciezka do pliku .exe
$daysValid = 365             # Waznosc certyfikatow
$timestampServer = "http://timestamp.digicert.com"
```

6.2 Uruchomienie skryptu

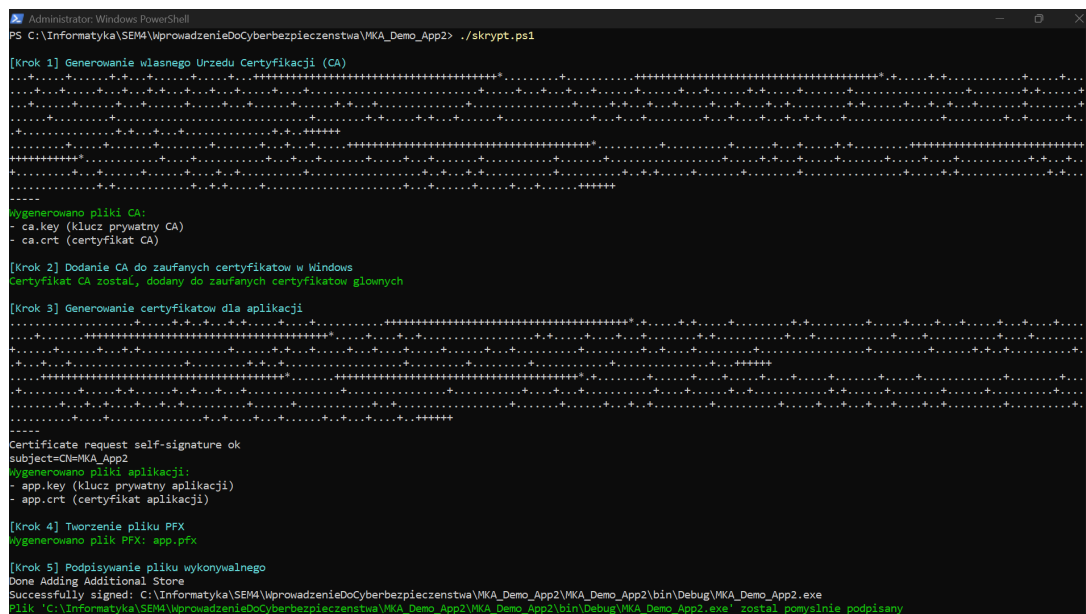
```
.\skrypt.ps1
```

Listing 7: Uruchomienie skryptu

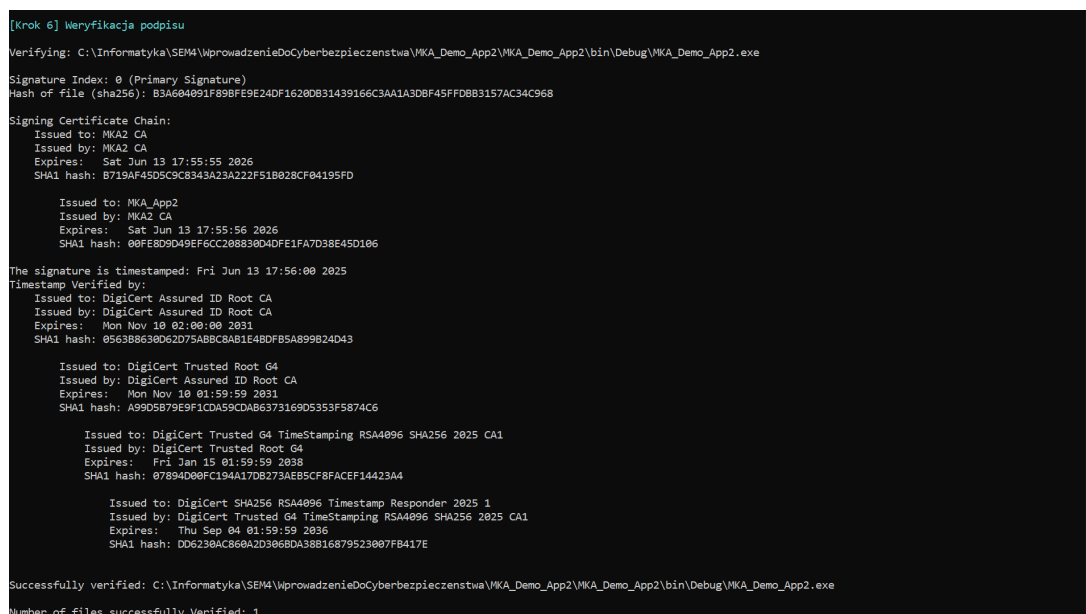
Wymagania:

- system Windows
- OpenSSL w systemowym PATH
- Windows SDK (dla signtool.exe)

6.3 Działanie skryptu



Rysunek 18: Wywołanie skryptu



Rysunek 19: Wywołanie skryptu

6.4 Bezpieczeństwo

Skrypt implementuje dobre praktyki bezpieczeństwa:

- Wymagane uruchomienie jako administrator
- Weryfikacja obecności wymaganych narzędzi (OpenSSL, signtool)
- Zabezpieczenie kluczy hasłami
- Walidacja każdego etapu procesu

6.5 Korzyści z automatyzacji

- Znaczne skrócenie czasu procesu
- Eliminacja błędów manualnych

7 Podsumowanie projektu

Projekt kompleksowo zrealizował proces podpisywania kodu w środowisku Windows, obejmujący:

Osiągnięcia

- **Infrastruktura certyfikacji**
 - Utworzenie własnego Urzędu Certyfikacji (CA) z rozszerzeniami X.509v3
 - Prawidłowa integracja z magazynem zaufanych certyfikatów Windows
 - Generacja certyfikatów aplikacji z wymaganymi rozszerzeniami do podpisywania kodu
- **Proces podpisywania**
 - Konwersja certyfikatów do formatu PFX gotowego do użycia w IDE
 - Implementacja podpisu cyfrowego w Visual Studio
 - Ręczna procedura podpisywania z wykorzystaniem `signtool.exe`
- **Zabezpieczenia czasowe**
 - Integracja z publicznym serwerem TSA (DigiCert)
 - Wdrożenie mechanizmu timestampingu SHA256
 - Weryfikacja długoterminowej ważności podpisów
- **Automatyzacja procesu**
 - Opracowanie skryptu PowerShell automatyzującego wszystkie etapy
 - Skrócenie czasu wykonania

Potwierdzone efekty

- Poprawna weryfikacja podpisu w systemie Windows
- Brak ostrzeżeń o "nieznanym wydawcy"
- Powtarzalność wyników dzięki zautomatyzowanemu procesowi

Wnioski

Projekt udowodnił możliwość budowy kompletnego rozwiązania do podpisywania kodu - od generacji infrastruktury kluczy, przez podpisywanie aplikacji, po weryfikację i automatyzację procesu.