Certificate Authority CBZ - Dokumentacja kodu

Piotr Pazdan, Maksymilian Oliwa

Styczeń 2026

II rok Cyberbezpieczeństwo, WIEiT

I Wprowadzenie

Niniejszy dokument stanowi udokumentowanie bazy kodu wchodzącego w skład głównego komponentu projektu, czyli programu `ca-cbz`. Program umożliwia generowanie żądań podpisu certyfikatu (ang. *certificate signing request*, *CSR*) oraz samych certyfikatów, zarówno tych podpisanych własnoręcznie (ang. *self-signed certificate*), jak i podpisanych za pomocą certyfikatu odrębnego urzędu certyfikacji.

Ze względu na obszerność standardu definiującego infrastrukturę klucza publicznego, program nie wspiera go w całości. Limit czasowy projektu zmusił nas do selektywnego podejmowania decyzji w kwestii wsparcia dla konkretnych schematów, szyfrów, rozszerzeń itd. Niemniej jednak zbiór obecnie zaimplementowanych mechanizmów w pełni wystarcza na efektywne użytkowanie i korzystanie z możliwości PKI.

II Infrastruktura klucza publicznego

Ważną kwestią jest istotność samego konceptu programu. Aby zrozumieć jego przeznaczenie, należy być zaznajomionym z technicznym opisem mechanizmów PKI, w szczególności wspomnianych wyżej certyfikatów. Zarysujmy zatem zestaw kroków, które muszą zostać spełnione, aby móc korzystać z bezpieczeństwa które oferuje standard. Za przykład posłuży obsługa protokołu HTTPS przez serwer sieciowy.

Protokół HTTPS służy weryfikacji (uwierzytelnieniu) serwera komunikującego się z klientem. Aby uwierzytelnienie było możliwe, należy stworzyć podmiot autoryzujący zaufane domeny odpowiednim certyfikatem. Rolę takiego podmiotu pełni urząd certyfikacji (ang. *certificate authority*), który w modelu łańcucha zaufania (ang. *chain of trust*) znajduje się na samej górze, tzn. jest zaufany z założenia.
Uwaga: W rzeczywistych systemach infrastruktura klucza publicznego obejmuje dodatkowo pośrednie urzędy certyfikacji (ang. *intermediate certificate authority*), których wiarygodność jest poświadczana przez urzędy certyfikacji wyższego szczebla. Zaufane z założenia są wyłącznie główne urzędy certyfikacji (ang. *root certificate authority*). W niniejszym wprowadzeniu świadomie posługujemy się uproszczonym modelem zaufania, w którym występuje jedynie główny urząd certyfikacji.

Urząd certyfikacji wystawia certyfikaty domenom, które spełnią następujące warunki:

1. dostarczą do urzędu żądanie podpisania certyfikatu
2. przedstawią dowód własności domeny widniejącej w żądaniu (odbywa się przeważnie poprzez wysłanie wiadomości z linkiem autoryzującym na parę adresów mailowych powiązanych z daną domeną)

Po pomyślnym przejściu procesu weryfikacji, certyfikat jest wystawiany na określony czas (z reguły parę miesięcy/lat).

Program `ca-cbz` oferuje funkcjonalności wspierające obie strony w tym procesie. Zapewnia narzędzia potrzebne właścicielowi domeny ubiegającemu się o certyfikat tj. generowanie żądań podpisu certyfikatu *CSR*, jak również niezbędne urzędowi w celu wystawiania certyfikatu - w tym generowanie certyfikatów podpisanych własnoręcznie jak i podpisanych przez zewnętrzny urząd certyfikacji.

Uwaga: Program **nie wspiera** generowania kluczy kryptograficznych - w tym celu zalecamy użycie gotowego i szeroko stosowanego oprogramowania *OpenSSL*.

III Formaty kodowania obiektów PKI

Dokumenty definiujące infrastrukturę klucza publicznego (RFC 5280, RFC 7468, itd.) zakładają, że pliki reprezentujące obiekty tej infrastruktury (klucze kryptograficzne, *CSR*, certyfikaty) będą reprezentowane w formacie *PEM* który jest powiązany z formatem *DER*, który to z kolei wykorzystuje format *ASN.1*. Krótki opis każdego z formatów:

- **ASN.1** - standard służący do opisu danych przeznaczonych do reprezentacji, kodowania lub dekodowania
- **DER** - jeden z wariantów kodowania ASN.1, tzn. konwersji abstrakcyjnych obiektów na ciąg bitowy (i *vice versa*)
- **PEM** - format ułatwiający transmisję danych zakodowanych w DER; dane są kodowane w Base64 oraz opatrzone nagłówkiem i stopką z odpowiednimi etykietami (ang. *labels*), np. `-----BEGIN CERTIFICATE-----`, `-----END CERTIFICATE-----`

IV Odgórny wzgląd na kod

Baza kodu programu `ca-cbz` składa się z zawartości katalogu `src` oraz pliku `Makefile`, służącego do kompilacji.
Wewnątrz katalogu `src` znajdziemy następujące elementy:

- `asn1` - katalog zawierający kod implementujący kodowanie i dekodowanie typów zdefiniowanych w standardzie ASN.1
- `encryption` - katalog definiujący prymitywy kryptograficzne, takie jak:
  (a) szyfr *AES-CBC* w odmiennych wariantach długości klucza
  (b) funkcja *HMAC* obliczająca kod uwierzytelnienia wiadomości
  (c) funkcja *PBKDF2* - szeroko stosowana odmiana funkcji z rodziny KDF (ang. *key derivation function*) służąca do wyznaczania klucza odpowiedniej długości na podstawie hasła
- `hash` - katalog z zawartymi algorytmami wyznaczania skrótów wiadomości; znajdują się tu różne warianty algorytmów z rodziny *SHA*
- `pkcs` - najbardziej obszerny katalog; zawiera kod odpowiedzialny za implementację mechanizmów powiązanych z *PKCS* (ang. *public-key cryptography standards*); znajduje się tu kod odpowiedzialny za zarządzanie kluczami kryptograficznymi, kodowanie obiektów PKCS zdefiniowanych w RFC 5280 oraz funkcje generujące i weryfikujące podpis cyfrowy.
- `tests` - katalog zawierający tzw. testy jednostkowe (ang. *unit tests*) weryfikujące sprawność różnych fragmentów kodu
- `utils` - katalog z funkcjonalonścią niesprecyzowanego przeznaczenia; głównie funkcje wspomagające sterowaniem wejścia-wyjścia, implementacja kodowania Base64 oraz funkcje związane z czyszczeniem buforów pamięciowych
- `main.cpp` - punkt startowy całego programu; znajduje się tu funkcja wejściowa `main` wraz z funkcjami pomocniczymi

V Szczegóły kodu

W tej sekcji przedstawiono szczegółowy opis wszystkich plików zawartych w projekcie.

5.1 `asn1/asn1.h`, `asn1/asn1.cpp`

W tych plikach zaimplementowana została funkcjonalność odpowiedzialna za obsługę standardu ASN.1.
Z racji tego, że obiekty ASN.1 tworzą złożone struktury, zdecydowaliśmy się na enkapsulację poszczególnych obiektów, efektywnie rozwiązując problem za pomocą programowania obiektowego.
By zrozumieć zawarte mechanizmy, warto przywołać strukturę pojedynczego obiektu według standardu ASN.1 - składa się on z pól `Type-Length-Value`, gdzie:

- `Type` - jednobajtowy typ obiektu (tag)
- `Length` - rozmiar obiektu (wartości), w bajtach
- `Value` - wartość obiektu o rozmiarze `Length`

Podstawowym elementem jest klasa `ASN1Object`, która pełni rolę kontenera na pojedynczy obiekt ASN.1. Zawiera następujące pola:

- `_tag` - tag reprezentowanego obiektu (`Type`)
- `_length` - długość wartości obiektu, w bajtach (`Length`)
- `_value` - kontener `std::vector` zawierający ciąg bajtów wartości obiektu
- `_children` - kontener `std::vector` zawierający inne instancje obiektu `ASN1Object` (dzieci), które mają być zinterpretowane jako zawartość obiektu rodzica
- `_encoded` - kontener `std::vector` zawierający zakodowaną w DER reprezentację obiektu - służy jako *cache* podczas wielokrotnego pobierania wartości zakodowanego obiektu.

Uwaga: od tego momentu słowa *obiekt* oraz *klasa* **nie będą** wykorzystywane naprzemiennie; poprzez *obiekt* należy rozumieć abstrakcyjny obiekt reprezentowany przez standard ASN.1 (struktura `Type-Length-Value`); poprzez *klasę* należy rozumieć klasę `ASN1Object`, służącą do reprezentacji obiektu ASN.1 w kodzie.

Klasa posiada również zdefiniowane konstruktory pobierające różne kombinacje powyższych pól. Przeciążane są również operatory `=` oraz `<`, co wykorzystywane jest przy sortowaniu obiektów (wymagane przez obiekt SET, wedle specyfikacji ASN.1). Do najważniejszych zdefiniowanych funkcji należą:

- `value` - zwraca referencję do bufora przetrzymującego wartość obiektu reprezentowanego przez instancję klasy
- `children` - zwraca referencję do bufora przetrzymującego *dzieci* danego obiektu reprezentowanego przez instancję klasy
- funkcje `encode` oraz `decode` - służą do zakodowania danej instancji klasy do formatu DER, oraz do zdekodowania formatu DER do instancji klasy o odpowiadającym stanie

Zaimplementowanych zostało również wiele poszczególnych typów obiektów ASN.1, (np. INTEGER czy OBJECT IDENTIFIER). Klasy reprezentujące te obiekty wykorzystują mechanizm dziedziczenia po klasie nadrzędnej `ASN1Object`, dodając jedynie specyficzne dla swojego obiektu konstruktory

oraz przeciążając funkcje `value` i `decode`, aby zwracały wartości bardziej adekwatne w kontekście obiektów które reprezentują. Przykładowo, funkcja `value` klasy `ASN1Integer` nie zwraca bufora przetrzymującego wartość obiektu, lecz konkretną wartość liczbową.

Klasy definiujące poszczególne obiekty są nazwane według schematu `ASN1{name}`, gdzie *name* to nazwa obiektu w standardzie ASN.1. Do zdefiniowanych w ten sposób klas dziedziczących po `ASN1Object` należą:

- `ASN1ObjectIdentifier`
- `ASN1Integer`
- `ASN1Sequence`
- `ASN1Null`
- `ASN1Boolean`
- `ASN1BitString`
- `ASN1OctetString`
- `ASN1Set`
- `ASN1UTCTime`
- `ASN1GeneralizedTime`

Wyjątek stanowi klasa `ASN1String`, która definiuje łańcuch znaków o dowolnym tagu obiektu - powstała po to, by móc reprezentować niewspierane warianty.
Uwaga: wymienione klasy stanowią reprezentacje tylko części z całości obiektów zdefiniowanych przez standard ASN.1; wedle standardu możliwe jest również definiowanie własnych typów obiektów, co jest stosowane przy tworzeniu obiektów PKCS.

W ramach obsługi obiektów czasu (`ASN1UTCTime`, `ASN1GeneralizedTime`) znajduje się tu również logika odpowiedzialna za konwersje dat i obliczanie znaczników czasu.

5.2 `encryption/aes.h`, `encryption/aes.cpp`

Zawierają kod odpowiedzialny za szyfry z rodziny AES. Ze względu na trudności związane z własnoręczną implementacją szyfru (napisanie efektywnej implementacji wymaga użycia języka assembler, aby móc wykorzystać instrukcje procesora wspierające schematy sprzętowo) zdecydowaliśmy się skorzystać z istniejących implementacji w bibliotece `libopenssl`. Zawartość plików sprowadza się więc do zdefiniowania typów poszczególnych rozmiarów klucza, oraz zarządzania pobranymi z biblioteki instancjami algorytmów.

5.3 `encryption/hmac.hpp`

Zawiera interfejs funkcji pseudolosowej (ang. *pseudo-random function*), który należy spełnić przy definiowaniu implementacji interfejsu. Znajduje się tu również szablon kryptograficznej funkcji HMAC, którego można użyć z różnymi wariantami zaimplementowanych funkcji skrótu.

5.4 encryption/kdf.hpp

Zawiera szablon funkcji kryptograficznej PBKDF2, której można użyć z różnymi wariantami funkcji pseudolosowej.

5.5 hash/sha.h, hash/sha.cpp

Przechowują interfejs funkcji skrótu oraz kod odpowiedzialny za zaimplementowane za jego pomocą funkcje; podobnie jak w przypadku szyfrów AES, zdecydowaliśmy się pobierać instancje potrzebnych algorytmów z biblioteki `libopenssl`.

5.6 pkcs/labels.h

Przechowuje zdefiniowane nagłówki oraz stopki wspieranych obiektów PKCS; są to:

- klucz prywatny (szyfrowany i nieszyfrowany)
- żądanie podpisu certyfikatu (*CSR*)
- certyfikat

5.7 pkcs/pkcs.h, pkcs/pkcs.cpp

Dwa największe pliki w całym projekcie. Definiują obiekty PKCS na podstawie struktur zdefiniowanych w RFC 5280, wykorzystując przy tym wyżej omówione klasy reprezentujące obiekty ASN.1. Dla każdej klasy zdefiniowane są funkcje pomocnicze zwracające poszczególne pola, oraz funkcje odpowiedzialne za kodowanie i dokodowanie w formacie DER. Zdefiniowano również wspierane algorytmy klucza prywatnego, schematy szyfrowania (ang. *encryption schemes*), funkcje KDF, HMAC, algorytmy szyfrowania (ang. *encryption algorithms*) oraz funkcje kodujące poszczególne obiekty i ich kombinacje do formatu zgodnego ze specyfikacją PKCS.

5.8 pkcs/private_key.h, pkcs/private_key.cpp

Definiują klasę `RSAPrivateKey`, używaną przez program do obsługi prywatnych kluczy kryptograficznych opartych na schemacie RSA. Klasa zawiera konstruktory pozwalające na odczyt istniejącego klucza z pliku; obsługiwany jest zarówno wariant nieszyfrowany, jak i klucz szyfrowany - wówczas przy otwieraniu należy podać hasło rozszyfrowujące zawartość klucza. W tym przypadku zaimplementowane algorytmy pozwalają na otwarcie klucza szyfrowanego metodą RSA, za pomocą szyfrowania *PBES2* z wykorzystaniem KDF (domyślny sposób szyfrowania klucza przez *OpenSSL*).

5.9 pkcs/public_key.h, pkcs/public_key.cpp

Zawierają klasę `RSAPublicKey`, definiującą klucz publiczny oparty na schemacie RSA. Wykorzystywana jest w obiektach PKCS posiadających sekcję opisującą klucz publiczny podmiotu.

5.10 pkcs/sign.h, pkcs/sign.cpp

Implementują logikę dotycząca mechanizmów podpisu cyfrowego tj. funkcje `RSASSA-PKCS1-v1_5` zawarte w RFC 8017. W kodzie zdefiniowane są:

- `RSASSA-PKCS1-v1_5_SIGN` - podpisuje określony bufor podanym kluczem
- `RSASSA-PKCS1-v1_5_VERIFY` - weryfikuje istniejący już podpis

5.11 `utils/base64.h`, `utils/base64.cpp`

Tu znajduje się implementacja kodowania Base64.

5.12 `utils/endianness.hpp`

Definiuje dwie funkcje konwertujące liczby całkowite do odpowiedniej repezetacji *Big Endian* lub *Little Endian*, w zależności od domyślnej kolejności bajtów systemu.

5.13 `utils/io.h`, `utils/io.cpp`

Zawierają funkcje pomocnicze zarządzające buforami wejścia-wyjścia. Umożliwiają odczyt i zapis na dysku lub z standardowego wejścia `stdin`.

5.14 `utils/security.hpp`

Zawiera funkcje związane z bezpieczeństwem. Są to m.in.:

- funkcje czyszczące bufor
- funkcja bezpiecznego odczytywania danych z pliku `secure_read_file` (`std::istream` w C++ nie pozwala na zarządzanie buforem wewnętrznym, więc istnieje ryzyko wycieku krytycznych danych)
- funkcja porównująca dwa bufory pamięci ciągłej w czasie stałym

5.15 `utils/utils.hpp`

Zawiera elementy pomocnicze wykorzystwane w różych miejscach programu, w tym:

- funkcje odpowiadające za wypisywanie czytelnie sformatowanych bajtów na ekran,
- funkcje zarządzające wyjątkami w C++
- naiwna implementacja dwukierunkowego słownika (ang. *bidirectional map*)

5.16 `main.cpp`

Punkt startowy całego programu. Zawiera funkcję `main` oraz funkcje pomocnicze sterujące przepływem programu:

- `handle_arguments` - przetwarza przekazane do programu argument
- `gen_self_signed_cert`, `gen_cert` - generują certyfikaty
- `gen_csr` - generuje żądania podpisu certyfikatu

Funkcje generujące wykorzystują implementacje obiektów PKCS zawarte w plikach `pkcs/pkcs.h` oraz `pkcs/pkcs.cpp`.