Bezpieczeństwo Systemów Komputerowych – Projekt

Kwalifikowany podpis elektroniczny

Raport

Michał Konieczny 188667

Konrad Czarnecki 193491

**1. Wstęp**

Celem projektu było zaprojektowanie oraz implementacja aplikacji umożliwiającej składanie kwalifikowanego podpisu elektronicznego zgodnie ze standardem PAdES. W ramach realizacji zadania należało stworzyć dwa główne komponenty: główną aplikację do podpisywania dokumentów PDF oraz aplikację pomocniczą służącą do generowania pary kluczy RSA i zabezpieczania klucza prywatnego. Klucz prywatny jest szyfrowany algorytmem AES przy użyciu hasła użytkownika i zapisywany na zewnętrznym nośniku USB, który emuluje sprzętowe narzędzie uwierzytelniające.

Podczas podpisywania dokumentu użytkownik musi podać PIN, którym deszyfrowany jest klucz prywatny z pendrive’a, a następnie wykorzystywany do podpisania wskazanego pliku PDF. Zgodnie z wymaganiami projektowymi, aplikacja automatycznie rozpoznaje podłączony nośnik USB i odczytuje z niego odpowiednie dane. Zaimplementowano również możliwość weryfikacji podpisu przez innego użytkownika, który na podstawie dokumentu i klucza publicznego może sprawdzić autentyczność podpisu oraz integralność dokumentu.

**2. Wykorzystane technologie**

W ramach projektu wykorzystano szereg technologii i bibliotek programistycznych, które umożliwiły bezpieczne generowanie, przechowywanie, używanie i weryfikację podpisu elektronicznego zgodnie ze standardem PAdES.

Język programowania: Python

Projekt został zrealizowany w języku Python ze względu na jego prostotę, czytelność oraz dostępność zaawansowanych bibliotek kryptograficznych.

Biblioteka PyCryptodome Zastosowana do realizacji operacji kryptograficznych takich jak:

* generowanie i import kluczy RSA (4096-bit),
* szyfrowanie klucza prywatnego algorytmem AES w trybie GCM,
* funkcje skrótu SHA-256 używane do generowania klucza AES z PIN-u.  
   PyCryptodome stanowi rozwinięcie i bezpieczną wersję biblioteki PyCrypto.

Biblioteka cryptography

Wykorzystywana do generowania certyfikatów X.509 oraz operacji związanych z obsługą kluczy w formatach PEM i DER. Biblioteka ta jest szeroko stosowana w aplikacjach wymagających wysokiego poziomu bezpieczeństwa.

PyPDF2 Kluczowa biblioteka do manipulacji plikami PDF, która umożliwia:

* dodawanie, usuwanie i kopiowanie stron PDF,
* dodawanie metadanych.

Doxygen Narzędzie do generowania dokumentacji kodu źródłowego. Umożliwia tworzenie przejrzystej dokumentacji funkcji i klas w projekcie, co jest istotne dla utrzymania przejrzystości i możliwości dalszego rozwoju projektu.

System plików i detekcja pendrive’a (moduł os)

W projekcie zastosowano proste podejście do detekcji nośnika USB na podstawie dostępnych liter dysków systemowych. Zostało to zrealizowane przy użyciu standardowej biblioteki os.

Wielowątkowość (threading)

Do monitorowania obecności pendrive’a z kluczem prywatnym użyto osobnego wątku działającego w tle, co pozwala na asynchroniczne reagowanie na podłączenie lub odłączenie urządzenia.

**3. Projekt interfejsu użytkownika**

**3.1 Aplikacja pomocnicza do generowania par kluczy**

Interfejs użytkownika aplikacji pomocniczej został zaprojektowany z myślą o maksymalnej prostocie i przejrzystości obsługi. Umożliwia on użytkownikowi szybkie wygenerowanie pary kluczy RSA oraz bezpieczne zapisanie klucza prywatnego na zewnętrznym nośniku USB.

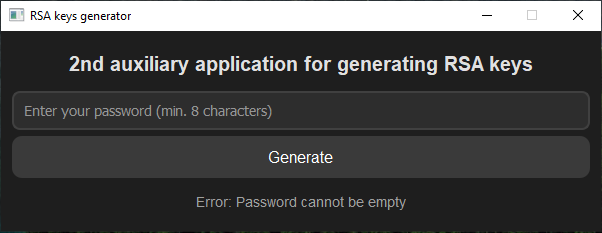
Główne elementy interfejsu to:

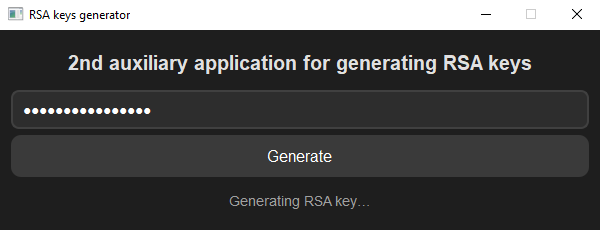
* Pole tekstowe (PIN): Użytkownik wprowadza w tym miejscu swój PIN, który służy do wygenerowania 256-bitowego klucza AES wykorzystywanego do szyfrowania klucza prywatnego RSA. Wprowadzone znaki są automatycznie maskowane (wyświetlane jako kropki), co zapewnia podstawowy poziom ochrony przed nieautoryzowanym podglądem. Dodatkowo zastosowano tekst pomocniczy (placeholder): „Enter your password (min. 8 characters)”, który informuje o minimalnych wymaganiach dotyczących długości hasła.
* Przycisk „Generate”: Rozpoczyna proces generowania 4096-bitowej pary kluczy RSA. W trakcie generowania użytkownik widzi dynamiczny komunikat informujący o stanie operacji, np. „Generating RSA key…”.
* Obsługa błędów: Jeżeli użytkownik spróbuje rozpocząć proces generowania kluczy bez uprzedniego wprowadzenia PIN-u, w dolnej części interfejsu pojawi się komunikat o błędzie: „Error: Password cannot be empty”.

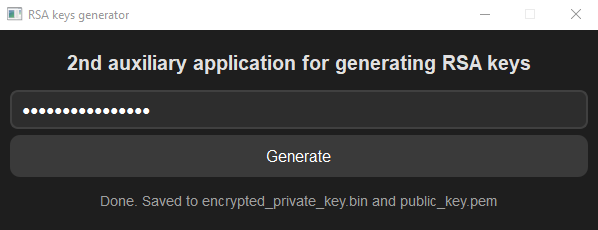
Taka walidacja danych wejściowych zapobiega przypadkowym błędom oraz poprawia bezpieczeństwo działania aplikacji.

* Komunikaty statusu: Po zakończeniu generowania kluczy pojawia się komunikat informacyjny:

„Done. Saved to encrypted\_private\_key.bin and public\_key.pem”, który potwierdza zapis zaszyfrowanego klucza prywatnego i klucza publicznego w odpowiednich plikach.







**3.2 Aplikacja główna do podpisu i weryfikacji dokumentu PDF**

Aplikacja główna została zaprojektowana jako graficzne narzędzie desktopowe umożliwiające użytkownikowi podpisywanie oraz weryfikację dokumentów PDF z wykorzystaniem podpisu elektronicznego w standardzie PAdES. Interfejs podzielony jest na dwie zakładki: „Sign Document” oraz „Verify Document”, które odpowiadają dwóm podstawowym trybom pracy aplikacji.

#### **Zakładka „Sign Document”**

W tej zakładce użytkownik wykonuje procedurę podpisu dokumentu PDF. Interfejs składa się z następujących elementów:

* Status pendrive’a

W górnej części aplikacji wyświetlany jest aktualny status wykrycia nośnika USB:

* + Jeśli nośnik nie jest podłączony, wyświetlany jest komunikat „No pendrive detected” (na czerwono).
  + Po podłączeniu pendrive’a, pojawia się informacja o znalezieniu dysku (np. „Pendrive found: H:\”) oraz o obecności pliku z zaszyfrowanym kluczem prywatnym (np. „Private key found: encrypted\_private\_key.bin”).
* Pole PIN-u

Użytkownik wprowadza tu hasło (PIN) wymagane do odszyfrowania klucza prywatnego z pendrive’a. Pole to automatycznie maskuje znaki.

* Przycisk „Select PDF to Sign”

Umożliwia wybór lokalnego pliku PDF, który ma zostać podpisany.

* Status wybranego pliku

Po wybraniu dokumentu, ścieżka do pliku jest wyświetlana pod przyciskiem wyboru.

* Przycisk „Sign Document”

Rozpoczyna procedurę podpisu. W zależności od poprawności podanych danych i obecności wymaganych plików, mogą wystąpić różne scenariusze:

* + Jeśli PIN jest nieprawidłowy, pojawia się komunikat „Error: Incorrect PIN” oraz okno błędu „Failed to sign document”.
  + W przypadku poprawnego PIN-u i prawidłowego przebiegu procesu, wyświetlany jest komunikat „Document signed successfully” wraz z lokalizacją zapisanego pliku (np. z dopiskiem \_signed.pdf).

#### **Zakładka „Verify Document”**

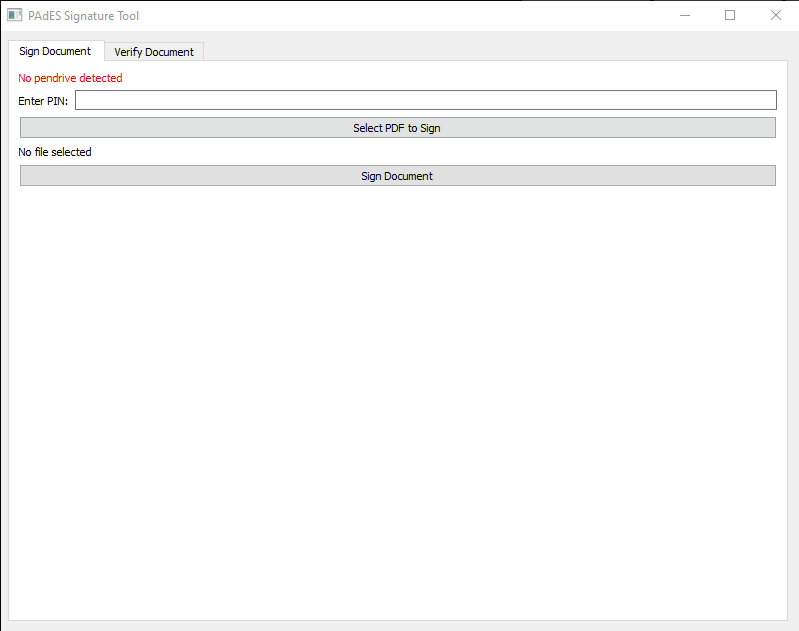
W drugiej zakładce możliwe jest zweryfikowanie podpisanego dokumentu PDF przy użyciu klucza publicznego. Interfejs zawiera:

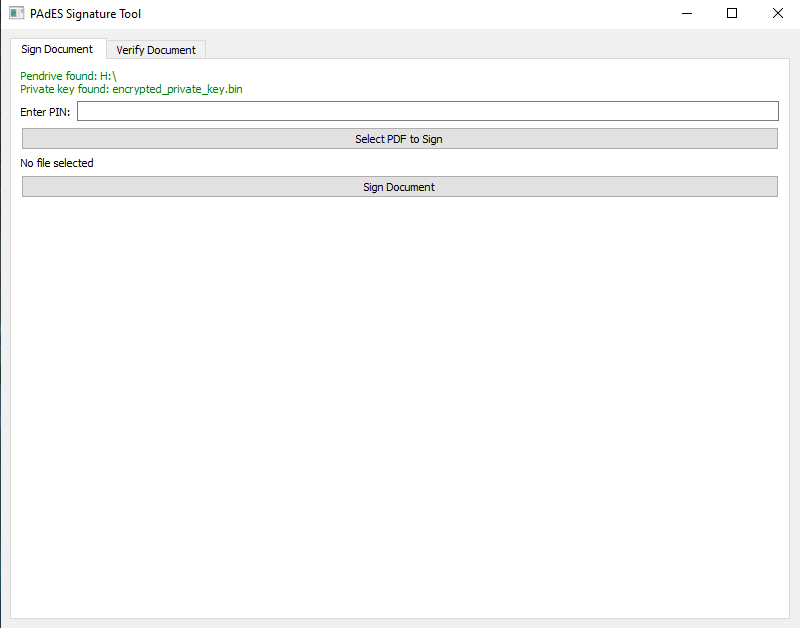
* Przycisk „Select PDF to Verify”  
   Służy do wyboru podpisanego dokumentu PDF.
* Przycisk „Select Public Key”

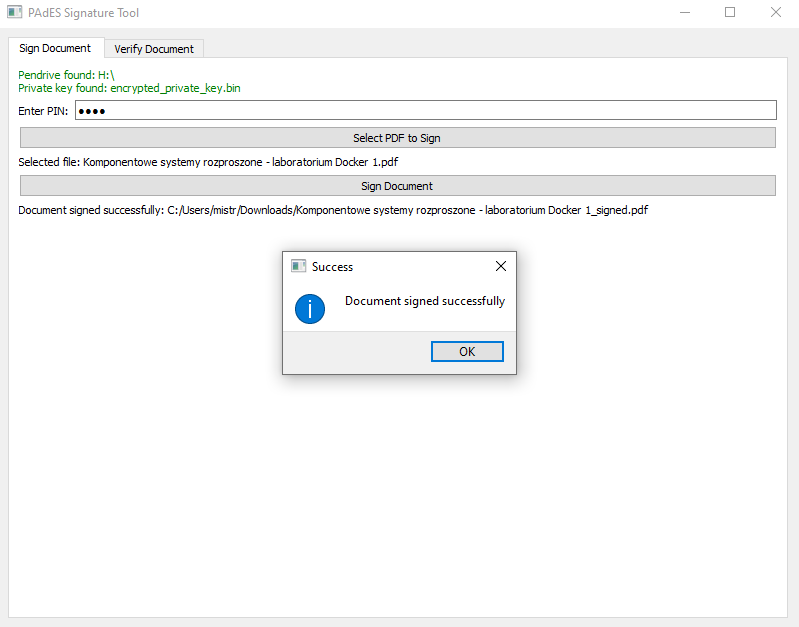
Umożliwia wskazanie pliku z kluczem publicznym użytkownika A (np. public\_key.pem), który był użyty do podpisu.

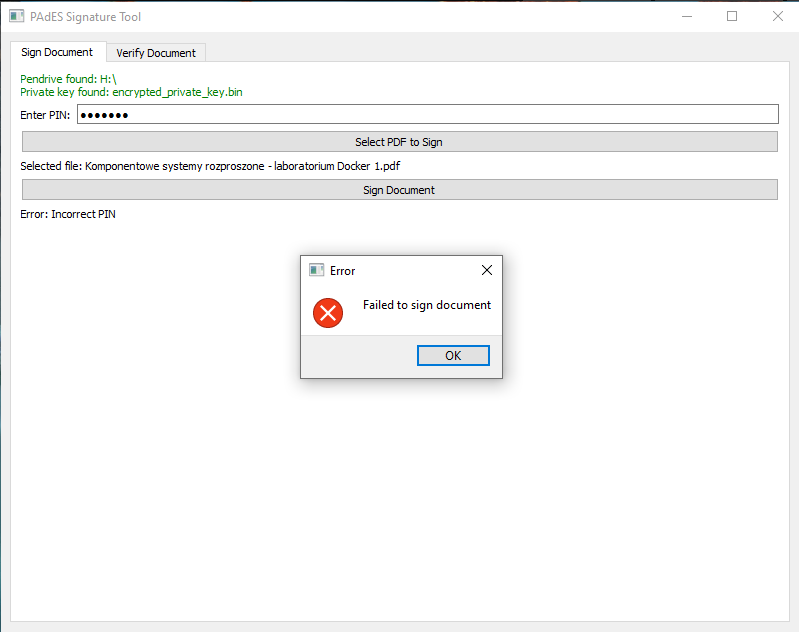
* Przycisk „Verify Document”

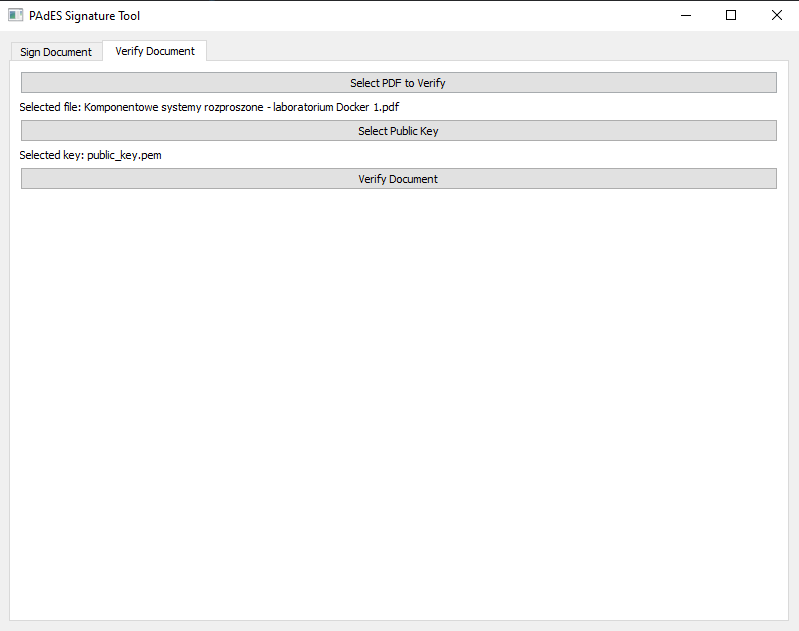
Rozpoczyna procedurę walidacji podpisu. Wyniki weryfikacji są prezentowane użytkownikowi w formie graficznej (np. komunikat o sukcesie lub błędzie).

****

****

****

****



**4. Opis funkcjonalności aplikacji**

**4.1. Główna aplikacja**

**Wykrywanie pendrive’a**

Funkcja stale przeszukuje litery dysków D, E, F, G i H, szuka pliku encrypted\_private\_key.bin i zapisuje ścieżkę.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

**Deszyfrowanie klucza prywatnego**

Funkcja deszyfruje klucz prywatny z pendrive z użyciem PIN-u podanego przez użytkownika.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

**Kopiowanie pliku PDF i tworzenie hasha**

Funkcja kopiuje strony oryginalnego pliku PDF i tworzy jego hash.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

**Podpis pliku PDF**

Funkcja podpisuje PDF hashem i datą wykonania podpisu.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

**Usunięcie metadanych**

Funkcja usuwa metadane z podanego pliku PDF, by odtworzyć hash oryginalnego pliku.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

**Weryfikacja podpisu**

Funkcja weryfikuje podpis PDF z użyciem hasha i klucza publicznego.



**4.2. Aplikacja pomocnicza**

**Hash PIN-u**

Funkcja hashuje PIN podany przez użytkownika.



**Klucz RSA**

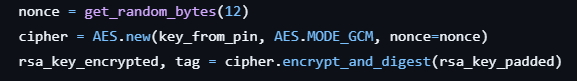
Funkcja generuje 4096-bitowe klucze RSA.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Zawartość wygenerowana przez AI może być niepoprawna.

**Szyfrowanie klucza prywatnego RSA**

Funkcja szyfruje klucz prywatny RSA z użyciem hasha PIN-u



**Sprawdzenie kluczy**

Funkcja sprawdza, czy klucze da się poprawnie zdekodować.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

**5. Scenariusze testowe**

**5.1 Aplikacja pomocnicza do generowania par kluczy**

**5.1.1 Cel: Sprawdzenie poprawnego uruchomienia aplikacji pomocniczej.**

Kroki:

1. Otwórz folder projektu w systemie plików.
2. Przejdź do katalogu *auxiliary\_app*.
3. Uruchom plik *auxiliary\_app.py* za pomocą interpretera Pythona.
4. Sprawdź, czy pojawiło się główne okno aplikacji z tytułem: „*2nd auxiliary application for generating RSA keys*”, polem do wpisania hasła oraz przyciskiem „*Generate*”.

Rezultat: Aplikacja uruchamia się poprawnie, wyświetlając kompletny i responsywny interfejs użytkownika.

**5.1.2 Cel: Sprawdzenie poprawnej obsługi błędu w przypadku braku hasła.**

Kroki:

1. Uruchom aplikację.
2. Nie wpisuj żadnego hasła w pole tekstowe.
3. Kliknij przycisk „*Generate*”.

Rezultat: Pojawia się komunikat błędu: „Error: Password cannot be empty”, a generowanie kluczy nie zostaje rozpoczęte.

**5.1.3 Cel: Wprowadzenie poprawnego hasła i generowanie kluczy**

Kroki:

1. Uruchom aplikację.
2. Wpisz hasło.
3. Kliknij przycisk „*Generate*”.
4. Przejdź do katalogu roboczego projektu.

Rezultat: Pojawia się komunikat: „Generating RSA key…”, a po zakończeniu: „Done. Saved to encrypted\_private\_key.bin and public\_key.pem”.

W katalogu roboczym pojawiają się dwa nowe pliki: *encrypted\_private\_key.bin*, który zawiera czytelny klucz w formacie PEM i *public\_key.pem* będący zaszyfrowanym plikiem binarnym.

**5.2 Aplikacja główna do podpisu i weryfikacji dokumentu PDF**

**5.2.1 Cel: Sprawdzenie poprawnego uruchomienia głównej aplikacji**

Kroki:

1. Otwórz folder projektu w systemie plików.
2. Przejdź do katalogu *main\_app*.
3. Uruchom plik *main\_app.py* za pomocą interpretera Pythona.
4. Poczekaj, aż załaduje się okno z zakładkami „*Sign Document*” oraz „*Verify Document*”.

Rezultat: Aplikacja uruchamia się poprawnie, prezentując interfejs z dwiema zakładkami oraz informacją o statusie pendrive’a.

**5.2.2 Cel: Weryfikacja zachowania aplikacji przy braku nośnika USB z kluczem**

Kroki:

1. Uruchom aplikację bez podłączonego pendrive’a.
2. Przejdź do zakładki „Sign Document”.

Rezultat: Wyświetlany jest komunikat w kolorze czerwonym:

„*No pendrive detected*”. Nie można przeprowadzić procesu podpisywania.

**5.2.3 Cel: Sprawdzenie poprawnej detekcji nośnika USB i zaszyfrowanego klucza**

Kroki:

1. Podłącz pendrive z plikiem „*encrypted\_private\_key.bin”*
2. Uruchom aplikację i przejdź do zakładki „Sign Document”.

Rezultat: Wyświetlany jest komunikat: „*Pendrive found: [litera dysku]*” oraz „Private key found: “*encrypted\_private\_key.bin*” (kolor zielony).

Aplikacja gotowa do podpisu.

**5.2.4 Cel: Sprawdzenie czy aplikacja poprawnie reaguje na fizyczne odłączenie i ponowne podłączenie nośnika USB w trakcie działania**

Kroki:

1. Uruchom aplikację i przejdź do zakładki „Sign Document”.
2. Podłącz pendrive’a z zaszyfrowanym kluczem („*encrypted\_private\_key.bin”).*
3. Poczekaj na komunikat: „Pendrive found: [litera dysku]” oraz „Private key found: encrypted\_private\_key.bin”.
4. Odłącz pendrive od komputera.
5. Obserwuj, czy status w aplikacji zmienia się na: „No pendrive detected” (na czerwono).
6. Podłącz pendrive do komputera.
7. Sprawdź, czy status automatycznie powraca do: „Pendrive found:…” oraz „Private key found…”.

Rezultat: Aplikacja dynamicznie reaguje na zmianę stanu nośnika USB bez konieczności ponownego uruchamiania. Użytkownik otrzymuje aktualne informacje w interfejsie, a podpisywanie dokumentów staje się ponownie dostępne po ponownym podłączeniu pendrive’a.

**5.2.5 Cel: Sprawdzenie obsługi nieprawidłowego hasła**

Kroki:

1. Uruchom aplikację i przejdź do zakładki „Sign Document”.
2. Podłącz pendrive’a z zaszyfrowanym kluczem („*encrypted\_private\_key.bin”).*
3. Wprowadź nieprawidłowy PIN w polu „Enter PIN”.
4. Kliknij „*Select PDF to Sign”* i wybierz plik PDF z dysku.
5. Kliknij „Sign Document”.

Rezultat: Wyświetlany jest komunikat: „Error: Incorrect PIN”, a następnie okno błędu: „Failed to sign document”. Podpis nie zostaje wykonany.

**5.2.6 Cel: Sprawdzenie reakcji aplikacji w przypadku próby podpisu dokumentu PDF bez wprowadzenia hasła (PIN-u)**

Kroki:

1. Uruchom aplikację i przejdź do zakładki „Sign Document”.
2. Podłącz pendrive'a z zaszyfrowanym kluczem („*encrypted\_private\_key.bin”).*
3. Kliknij „*Select PDF to Sign”* i wybierz plik PDF z dysku.
4. Nie wpisuj żadnego PIN-u w polu „Enter PIN”.
5. Kliknij przycisk „Sign Document”.

Rezultat: Pojawia się okno dialogowe z ostrzeżeniem: „Please enter PIN” i ikona wykrzyknika (warning). Proces podpisu nie zostaje rozpoczęty, a użytkownik musi uzupełnić pole PIN, aby kontynuować.

**5.2.7 Cel: Sprawdzenie pełnego procesu podpisywania dokumentu z poprawnym PIN-em**

Kroki:

1. Uruchom aplikację i przejdź do zakładki „Sign Document”.
2. Podłącz pendrive'a z zaszyfrowanym kluczem („*encrypted\_private\_key.bin”).*
3. Kliknij „*Select PDF to Sign”* i wybierz plik PDF z dysku.
4. Wprowadź poprawny PIN.
5. Kliknij „Sign Document”.

Rezultat: Wyświetlany jest komunikat: „Document signed successfully: [ścieżka do pliku]” oraz okno sukcesu. Podpisany elektronicznie dokument zostaje zapisany w folderze, z którego pochodził oryginalny plik PDF. Nowy plik PDF powinien mieć nazwę zakończoną na \_signed.pdf.

**5.2.8 Cel: Sprawdzenie, czy podpisany dokument może zostać poprawnie zweryfikowany**

Kroki:

1. Uruchom aplikację i przejdź do zakładki „Verify Document”.
2. Kliknij „Select PDF to Verify” i wskaż podpisany plik (z końcówką \_signed.pdf).
3. Kliknij „Select Public Key” i wybierz plik *public\_key.pem*.
4. Kliknij „Verify Document”.

Rezultat: Podpis zostaje poprawnie zweryfikowany.

Wyświetlany jest pozytywny komunikat o zgodności podpisu.

**5.2.9 Cel: Weryfikacja niepodpisanego lub zmodyfikowanego dokumentu**

Kroki:

1. Otwórz podpisany w scenariuszu testowym 5.2.8 dokument i dokonaj jego edycji (np. dodaj znak).
2. Zapisz plik.
3. Uruchom aplikację i przejdź do zakładki „Verify Document”.
4. Kliknij „Select PDF to Verify” i wskaż zmodyfikowany podpisany plik.
5. Kliknij „Select Public Key” i wybierz plik *public\_key.pem*.
6. Kliknij „Verify Document”.

Rezultat: Podpis zostaje uznany za niepoprawny.

Aplikacja informuje o błędzie walidacji lub niespójności danych.

**6. Bibliografia**

1. PyCryptodome Documentation – *Python Cryptographic Library for AES, RSA, SHA (<https://pycryptodome.readthedocs.io/>)*

2. cryptography.io Documentation – *Cryptography library for Python* *(<https://cryptography.io/>)*

3. Doxygen Documentation Generator – *Tool for automatic source code documentation (<https://www.doxygen.nl/>)*

4. Python Software Foundation – *The Python Language Reference, v3.11 (<https://docs.python.org/3/>)*

5. Stack Overflow – *Various discussions on cryptographic implementations and GUI troubleshooting (<https://stackoverflow.com/>)*