Bezpieczeństwo Systemów Komputerowych – Projekt

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 PyQt5
Biblioteka GUI wykorzystana do stworzenia interfejsu graficznego aplikacji głównej. Obsługuje zakładki, przyciski, pola tekstowe, komunikaty oraz dynamiczne aktualizowanie statusów (np. podłączenie pendrive'a, powodzenie operacji).

Biblioteka hashlib
Wykorzystywana do generowania skrótów SHA-256:

- z treści dokumentu PDF (podpis/weryfikacja)
- z PIN-u użytkownika (jako klucz do AES)

Biblioteka PyCryptodome
PyCryptodome stanowi rozwinięcie i bezpieczną wersję biblioteki PyCrypto.
Zastosowana do realizacji operacji kryptograficznych takich jak:

- generowanie i import kluczy RSA (4096-bit),
- szyfrowanie klucza prywatnego algorytmem AES w trybie ECB,
- funkcje skrótu SHA-256 używane do generowania klucza AES z PIN-u.

Biblioteka 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.

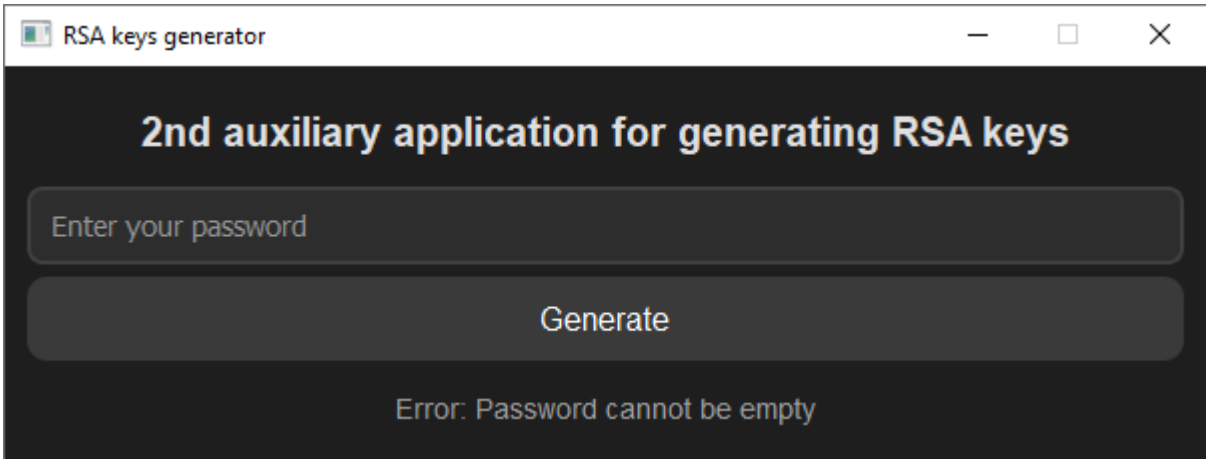
Główne elementy interfejsu to:

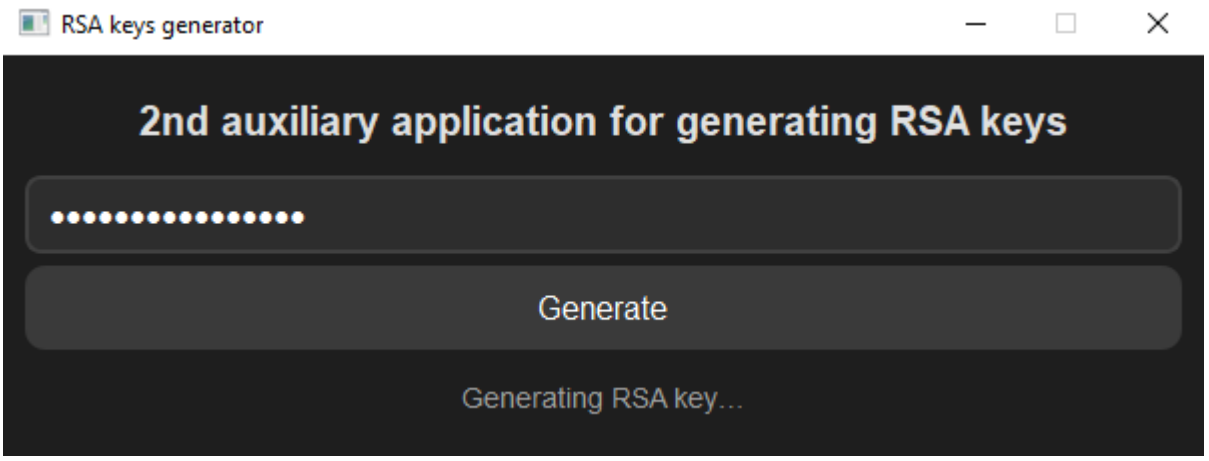
- Pole tekstowe (PIN): Użytkownik wprowadza w tym miejscu swój PIN, który służy do wygenerowania 256-bitowego klucza AES wykorzystywanego do szyfrowania klucza prywatnego RSA. Wprowadzone znaki są automatycznie maskowane (wyświetlane jako kropki), co zapewnia podstawowy poziom ochrony przed nieautoryzowanym podglądem.

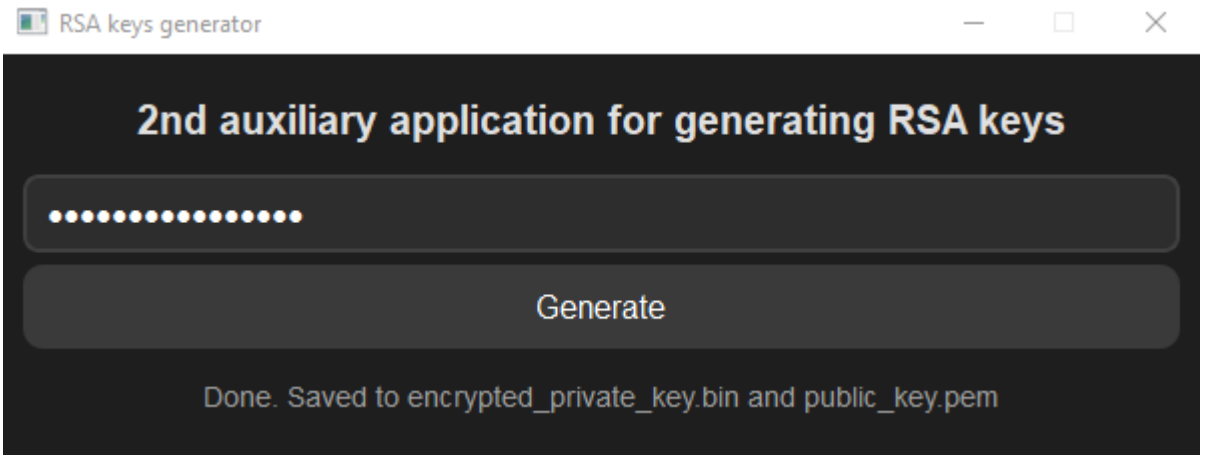
- Przycisk "Generate": Rozpoczyna proces generowania 4096-bitowej pary kluczy RSA. W trakcie generowania użytkownik widzi dynamiczny komunikat informujący o stanie operacji, np. "Generating RSA key…".

- 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.


RSA keys generator
2nd auxiliary application for generating RSA keys
Enter your password
Generate
Error: Password cannot be empty


RSA keys generator
2nd auxiliary application for generating RSA keys
Generate
Generating RSA key...


RSA keys generator
2nd auxiliary application for generating RSA keys
Generate
Done. Saved to encrypted_private_key.bin and public_key.pem

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).

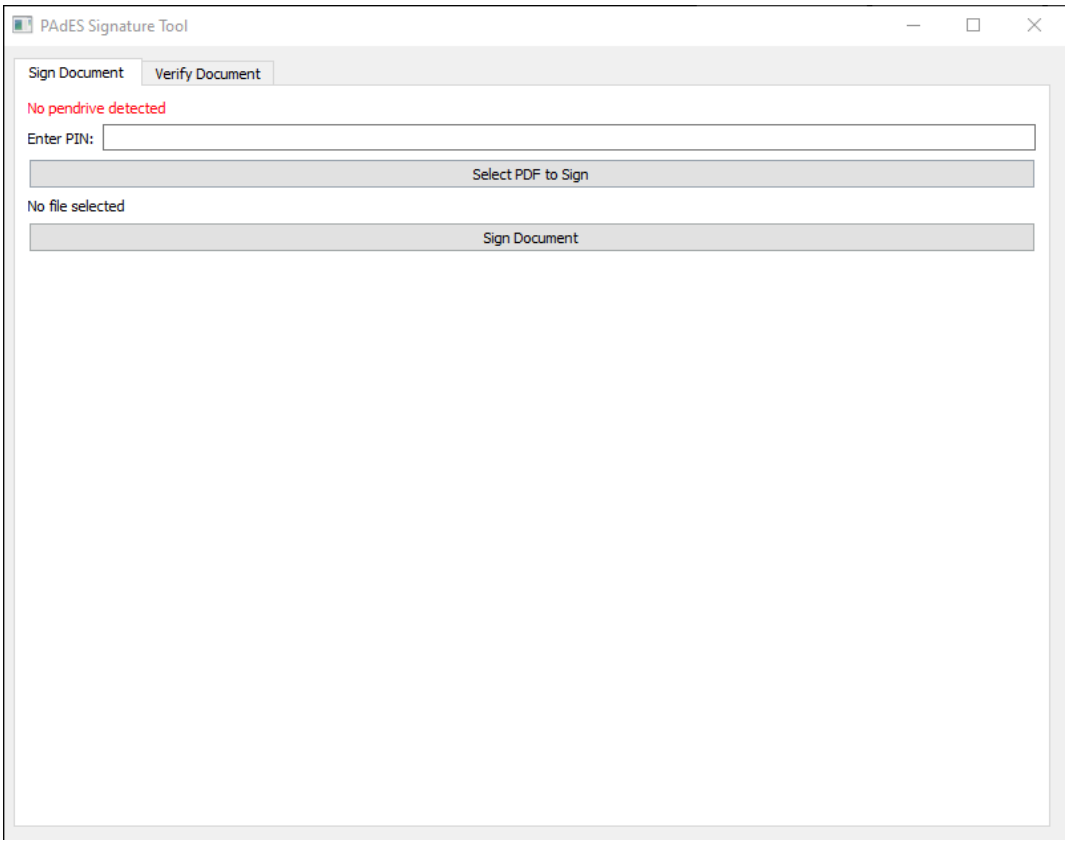
Zakładka "Verify Document"

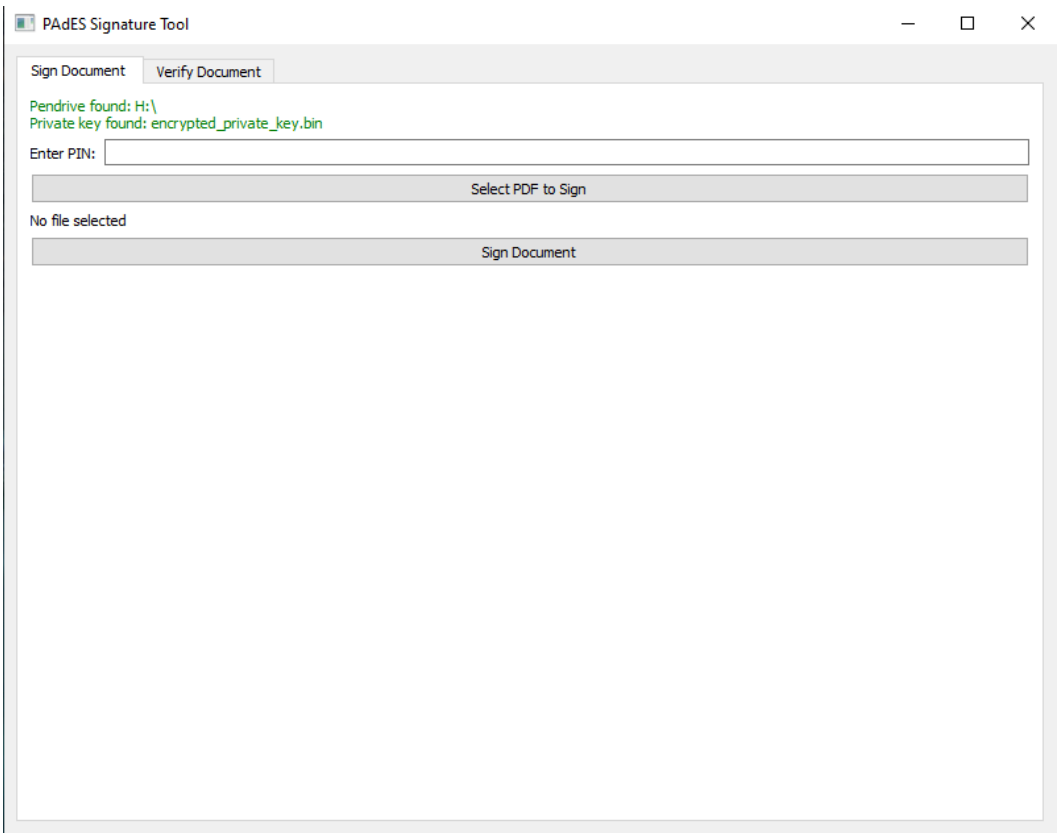
W drugiej zakładce możliwe jest zweryfikowanie podpisanego dokumentu PDF przy użyciu klucza publicznego. Interfejs zawiera:

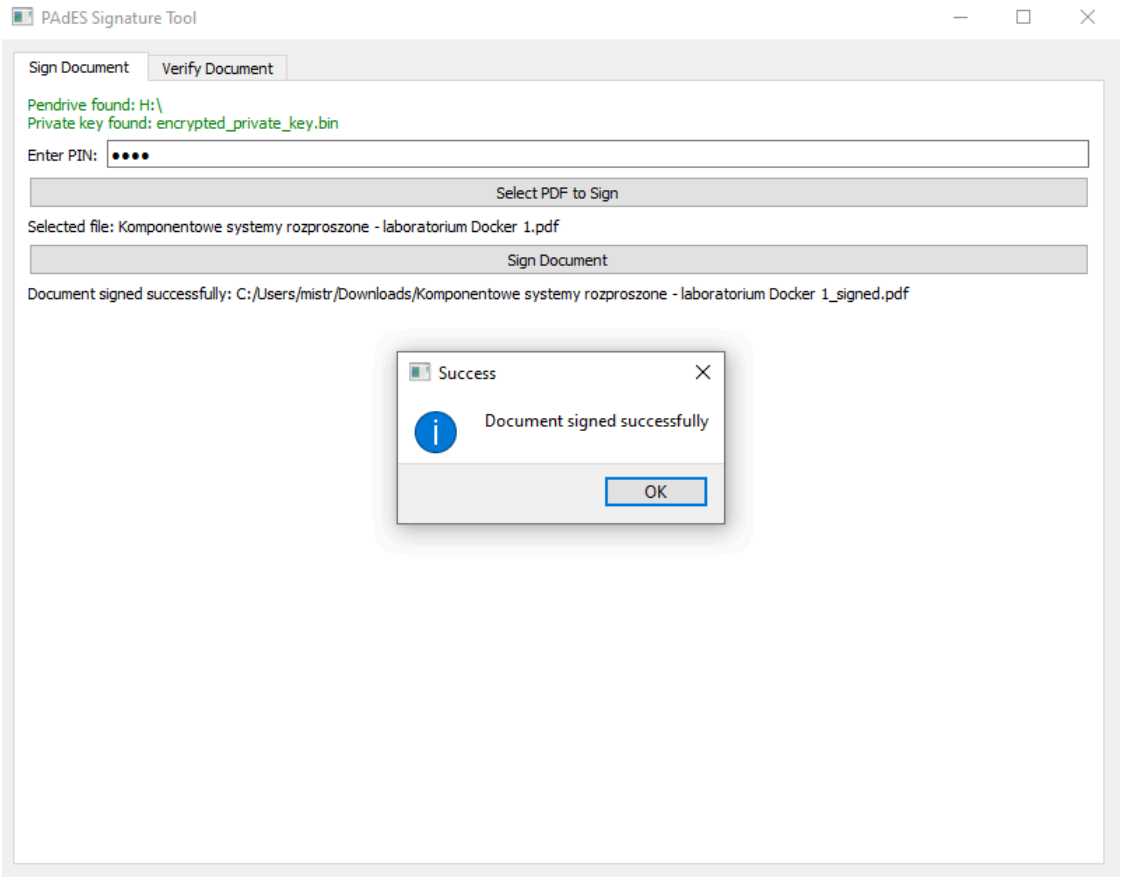
- Przycisk "Select PDF to Verify"
  Służy do wyboru podpisanego dokumentu PDF.

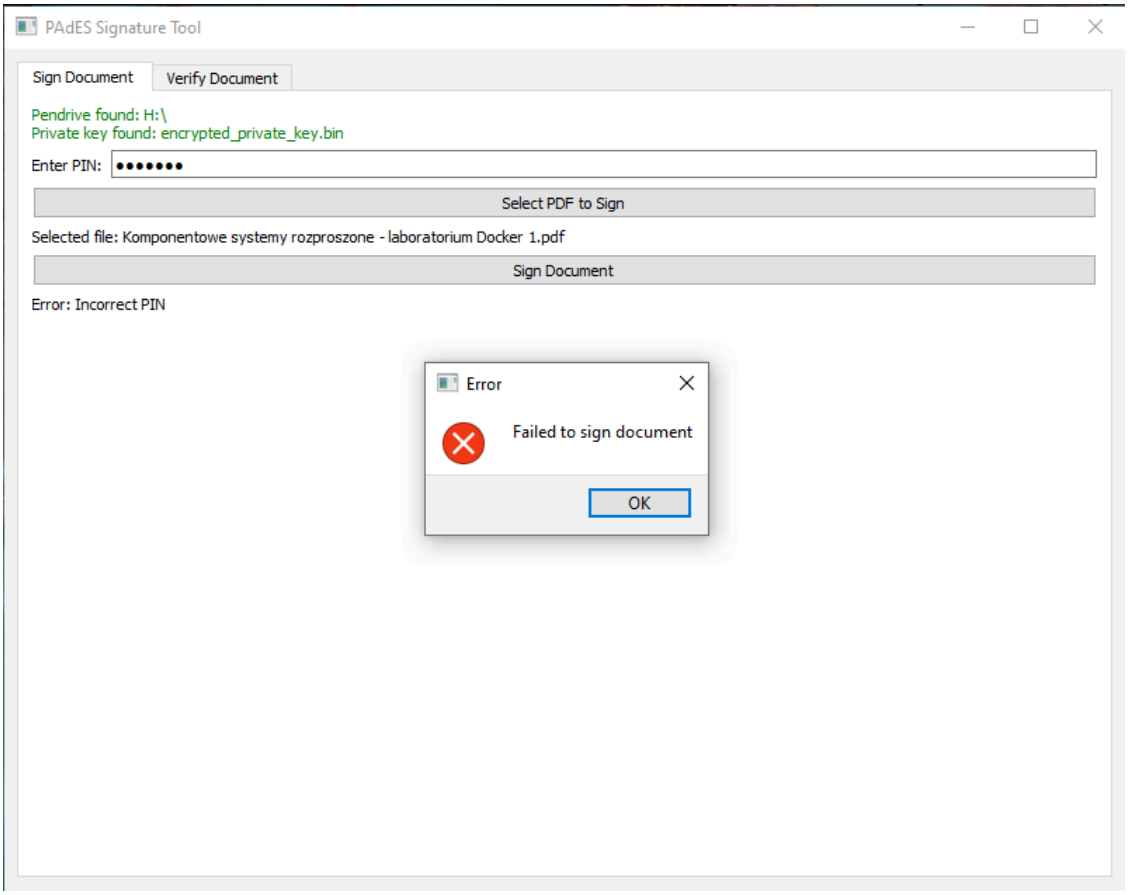
- Przycisk "Select Public Key"
  Umożliwia wskazanie pliku z kluczem publicznym użytkownika A (np. public_key.pem), który był użyty do podpisu.

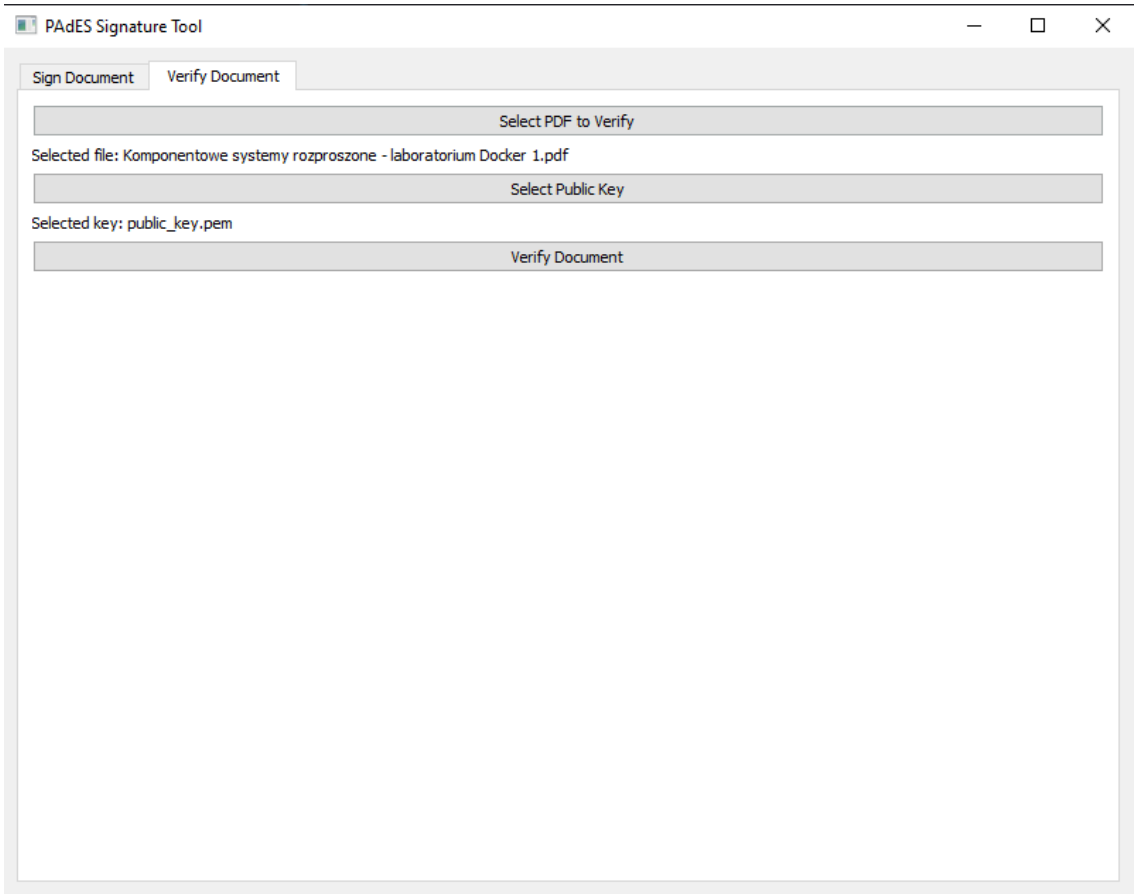
- Przycisk "Verify Document"
  Rozpoczyna procedurę walidacji podpisu. Wyniki weryfikacji są prezentowane użytkownikowi w formie graficznej (np. komunikat o sukcesie lub błędzie).

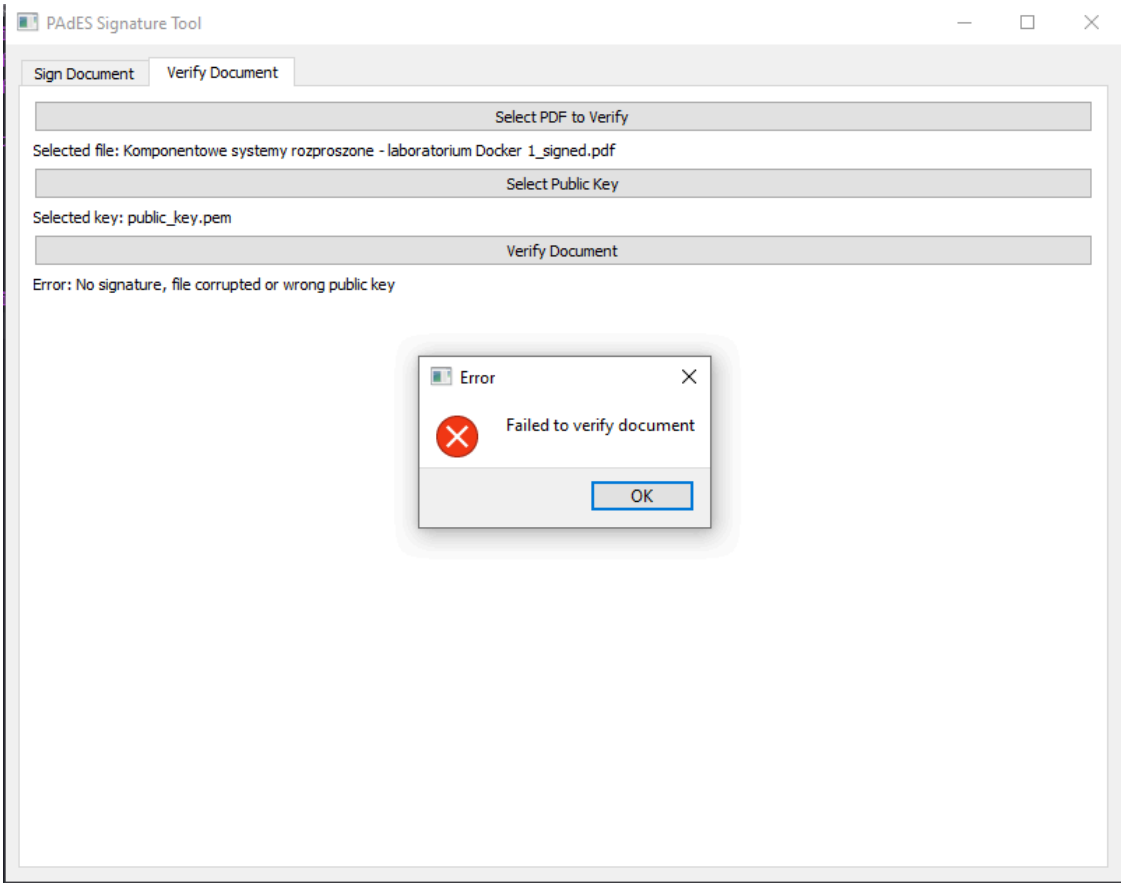


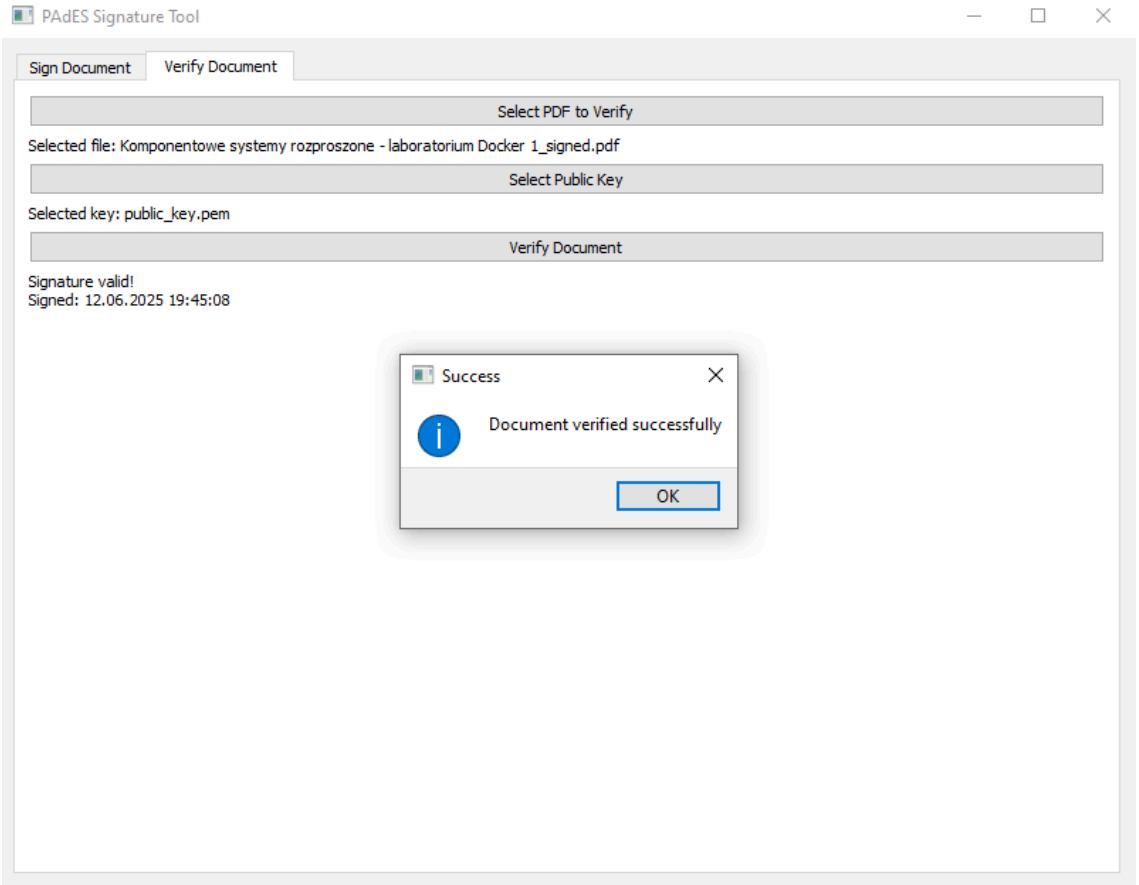

PAdES Signature Tool
Sign Document
Verify Document
Pendrive found: H:\
Private key found: encrypted_private_key.bin
Enter PIN:
Select PDF to Sign
No file selected
Sign Document


PAdES Signature Tool
Sign Document
Verify Document
Pendrive found: H:\
Private key found: encrypted_private_key.bin
Enter PIN:
Select PDF to Sign
Selected file: Komponentowe systemy rozproszone - laboratorium Docker 1.pdf
Sign Document
Document signed successfully: C:/Users/mistr/Downloads/Komponentowe systemy rozproszone - laboratorium Docker 1_signed.pdf
Success
Document signed successfully
OK


PAdES Signature Tool
Sign Document
Verify Document
Pendrive found: H:\
Private key found: encrypted_private_key.bin
Enter PIN: •••••••
Select PDF to Sign
Selected file: Komponentowe systemy rozproszone - laboratorium Docker 1.pdf
Sign Document
Error: Incorrect PIN
Error
Failed to sign document
OK


PAdES Signature Tool
Sign Document
Verify Document
Select PDF to Verify
Selected file: Komponentowe systemy rozproszone - laboratorium Docker 1.pdf
Select Public Key
Selected key: public_key.pem
Verify Document


PAdES Signature Tool
Sign Document
Verify Document
Select PDF to Verify
Selected file: Komponentowe systemy rozproszone - laboratorium Docker 1_signed.pdf
Select Public Key
Selected key: public_key.pem
Verify Document
Error: No signature, file corrupted or wrong public key
Error
Failed to verify document
OK


PAdES Signature Tool
Sign Document
Verify Document
Select PDF to Verify
Selected file: Komponentowe systemy rozproszone - laboratorium Docker 1_signed.pdf
Select Public Key
Selected key: public_key.pem
Verify Document
Signature valid!
Signed: 12.06.2025 19:45:08
Success
Document verified successfully
OK

4. Opis funkcjonalności aplikacji

4.1. Główna aplikacja

Wykrywanie pendrive'a

Aplikacja automatycznie przeszukuje wybrane litery dysków (D, E, F, G, H) w celu odnalezienia pliku encrypted_private_key.bin. Po wykryciu pliku zapisuje ścieżkę do nośnika oraz klucza prywatnego.

```
def check_for_pendrive(self):
    drives = ['D:\\', 'E:\\', 'F:\\', 'G:\\', 'H:\\']
    for drive in drives:
        if os.path.exists(drive):
            key_path = os.path.join(drive, "encrypted_private_key.bin")
            if os.path.exists(key_path):
                try:
                    with open(key_path, 'rb') as f:
                        data = f.read()
                        if len(data) > 28:
                            self.pendrive_path = drive
                            self.private_key_path = key_path
                            return True
                except:
                    continue
    self.pendrive_path = None
    self.private_key_path = None
    return False
```

Deszyfrowanie klucza prywatnego

Po podaniu PIN-u przez użytkownika, aplikacja generuje z niego 256-bitowy klucz AES przy użyciu funkcji skrótu SHA-256. Następnie odczytuje zaszyfrowany klucz prywatny z pliku na pendrive'ie i odszyfrowuje go przy użyciu trybu AES-ECB. Ostatecznie usuwane jest dopełnienie bajtowe (padding), a odszyfrowany klucz wykorzystywany jest do podpisywania dokumentu.

```
key_from_pin = SHA256.new(pin.encode()).digest()
with open(self.private_key_path, 'rb') as f:
    encrypted_key = f.read()
cipher = AES.new(key_from_pin, AES.MODE_ECB)
decrypted_padded_key = cipher.decrypt(encrypted_key)
decrypted_key = unpad(decrypted_padded_key, AES.block_size)
RSA.import_key(decrypted_key)
```

Kopiowanie pliku PDF i tworzenie hasha

Aplikacja odczytuje wszystkie strony z oryginalnego pliku PDF i zapisuje je do strumienia pamięci (BytesIO) za pomocą obiektu PdfWriter. Następnie generowany jest skrót SHA-256 całej zawartości dokumentu. Wygenerowany hash jest używany do podpisania dokumentu.

```
pdf_writer.add_metadata({
    '/DigitalSignature': private_key.sign(hash, padding.PKCS1v15(), hashes.SHA256()).hex(),
    '/DigitalSignatureTimestamp': datetime.now().strftime("%d.%m.%Y %H:%M:%S")
})
```

Podpis pliku PDF

Funkcja dodaje do metadanych dokumentu cyfrowy podpis, który powstaje poprzez zaszyfrowanie hasha dokumentu kluczem prywatnym przy użyciu algorytmu RSA z paddingiem PKCS#1 v1.5 oraz funkcją skrótu SHA-256. Dodatkowo, do metadanych dodawany jest timestamp podpisu dokumentu.

```
    pdf_writer.add_metadata({
        '/DigitalSignature': private_key.sign(hash, padding.PKCS1v15(), hashes.SHA256()).hex(),
        '/DigitalSignatureTimestamp': datetime.now().strftime("%d.%m.%Y %H:%M:%S")
    })
```

Usunięcie metadanych

Aby możliwe było poprawne odtworzenie hasha podpisanego dokumentu podczas jego weryfikacji, funkcja usuwa z metadanych pliku PDF pola związane z podpisem. Dzięki temu pomijane są dynamiczne informacje, które mogłyby wpłynąć na wynik funkcji skrótu. Pozwala to na porównanie oryginalnego hasha dokumentu z hashem odszyfrowanym z podpisu i weryfikację integralności pliku.

```
pdf_writer = PdfWriter()
for a in PdfReader(pdf_path).pages: pdf_writer.add_page(a)
temp_metadata = PdfReader(pdf_path).metadata.copy()
for i in data:
    if i in temp_metadata: del temp_metadata[i]
pdf_writer.add_metadata(temp_metadata)

signed_stream = io.BytesIO()
pdf_writer.write(signed_stream)
signed_stream.seek(0)
hash = hashlib.sha256(signed_stream.read()).digest()
```

Weryfikacja podpisu

Funkcja verify() porównuje hash wygenerowany z dokumentu z podpisem odszyfrowanym z pliku PDF. Jeśli dane te są zgodne, podpis uznawany jest za autentyczny.

```
public_key.verify(bytes.fromhex(pdf_data[0]), hash, padding.PKCS1v15(), hashes.SHA256())
```

4.2. Aplikacja pomocnicza

Hash PIN-u

W aplikacji pomocniczej wprowadzony przez użytkownika PIN jest przekształcany do postaci 256-bitowego skrótu przy użyciu algorytmu SHA-256. Wygenerowany hash stanowi klucz do szyfrowania klucza prywatnego.

```
key_from_pin = sha256(pin.encode()).digest()
```

Klucz RSA

Funkcja generuje parę kluczy RSA o długości 4096 bitów. Klucz prywatny i publiczny są eksportowane w formacie binarnym.

```
rsa_key_pair = RSA.generate(4096)
rsa_private_key = rsa_key_pair.export_key()
rsa_public_key = rsa_key_pair.publickey().export_key()
rsa_key_padded = pad(rsa_private_key, AES.block_size)
```

Szyfrowanie klucza prywatnego RSA

Klucz prywatny RSA, po uprzednim dopełnieniu, jest szyfrowany algorytmem AES w trybie ECB z użyciem klucza wygenerowanego z PIN-u użytkownika (hash SHA-256).

```
progress_callback.emit("Encrypting RSA key…")
cipher = AES.new(key_from_pin, AES.MODE_ECB)
rsa_key_encrypted = cipher.encrypt(rsa_key_padded)
```

Sprawdzenie kluczy

Funkcja sprawdza, czy zaszyfrowany klucz prywatny RSA można poprawnie odszyfrować przy użyciu podanego PIN-u. W tym celu tworzony jest szyfr AES w trybie ECB i podejmowana jest próba odszyfrowania. Jeśli operacja zakończy się niepowodzeniem, zgłaszany jest wyjątek i wypisywany komunikat o błędzie.

```
try:
    decrypt_cipher = AES.new(key_from_pin, AES.MODE_ECB)
    decrypted_padded = decrypt_cipher.decrypt(rsa_key_encrypted)
    decrypted_key = unpad(decrypted_padded, AES.block_size)
    print("Verification successful - key can be decrypted")
except Exception as e:
    print(f"Warning: Verification failed - {str(e)}")
```

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 został 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 został poprawnie zweryfikowany.
Wyświetlany jest komunikat: "Signature valid! Signed: [dd.mm.yyyy hh:mm:ss]" oraz okno sukcesu.

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 poprzez komunikat: "Error: No signature, file corrupted or wrong public key".

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/)*

8. Link do repozytorium GitHub

https://github.com/kmonieczny/Tool-for-Emulating-the-PAdES-Qualified-Electronic-Signature/tree/main