
Security of Computer Systems

Project Report

Authors:
Henryk Wołek 193399
Adam Chabraszewski 193373

Version: 1.0

*** * * REMOVE * * ***

During realization of the project please extend the document, do not create separate documents control and submission term.

*** * * REMOVE * * ***

Versions

Version	Date	Description of changes
1.0	08.04.2025	Creation of the document
1.1

1. Project – Final term

1.1 Opis projektu

Głównym celem projektu jest stworzenie desktopowej aplikacji służącej do podpisywania i weryfikowania integralności plików PDF. Został on napisany w języku Python i jest podzielona na trzy części: pierwsza generuje klucze RSA (4096 bitów) i zapisuje zaszyfrowany klucz prywatny na pendrive, zabezpieczony PIN-em; druga służy do podpisywania PDF-ów przy użyciu tego klucza z pendrive'a; trzecia umożliwia sprawdzenie, czy podpis jest ważny, używając klucza publicznego. Pod spodem użyte zostały biblioteki cryptography, pycryptodome (do AES i SHA-256) i win32api do wykrywania USB na Windowsie.

1.2 Code Description

Każda z osobnych części projektu funkcjonuje jako osobna aplikacja i znajduje się w osobnym pliku

1) main.py

Ten program służy do wygenerowania pary kluczy RSA (prywatnego oraz publicznego) oraz zapisania ich na pendrive zablokowanego kodem PIN. Składa się on z klasy USBKeyApp posiadającej wszelkie potrzebne metody

main.USBKeyApp Class Reference

Aplikacja GUI do generowania pary kluczy RSA i zapisywania ich na pendrive. More...

Inheritance diagram for main.USBKeyApp:

```
graph TD
    Tk[Tk.Tk] --> USBKeyApp[main.USBKeyApp]
```

Public Member Functions

<code>__init__(self)</code>	Konstruktor klasy USBKeyApp.
<code>log_message(self, message)</code>	Loguje wiadomość do okna oraz do konsoli.
<code>get_drive_letters(self)</code>	Zwraca listę aktualnie podłączonych dysków.
<code>encrypt_private_key(self, private_key_bytes, pin)</code>	Szyfruje prywatny klucz RSA za pomocą PIN-u użytkownika.
<code>save_public_key(self, private_key)</code>	Zapisuje publiczny klucz RSA do pliku PEM.
<code>handle_usb_insertion(self, drive_letter)</code>	Obsługuje włożenie pendrive'a i generuje klucz RSA.
<code>poll_for_usb(self)</code>	Wątek do monitorowania podłączenia pendrive'a.

Public Attributes

<code>log = scrolledtext.ScrolledText(self, wrap=tk.WORD)</code>
<code>previous_drives = self.get_drive_letters()</code>
<code>poll_usb_thread = threading.Thread(target=self.poll_for_usb, daemon=True)</code>

Dwie funkcje obsługujące nośnik USB to `poll_for_usb()` oraz `handle_usb_insertion()`

• handle_usb_insertion()

`main.USBKeyApp.handle_usb_insertion (self, drive_letter)`

Obsługuje włożenie pendrive'a i generuje klucz RSA.

Parameters

`drive_letter` Nazwa dysku, na którym wykryto pendrive'a.

• poll_for_usb()

`main.USBKeyApp.poll_for_usb (self)`

Wątek do monitorowania podłączenia pendrive'a.

Dwie funkcje odpowiedzialne za stworzenie klucza publicznego oraz prywatnego to `encrypt_private_key()` oraz `save_public_key()`

```
◆ encrypt_private_key()
main.USBKeyApp.encrypt_private_key ( self,
                                     private_key_bytes,
                                     pin )
```

Szyfruje prywatny klucz RSA za pomocą PIN-u użytkownika.

Parameters

private_key_bytes Bajty klucza prywatnego w formacie PEM.
pin 4-cyfrowy PIN wprowadzony przez użytkownika.

Returns

Szyfrowane bajty klucza prywatnego.

```
◆ save_public_key()
main.USBKeyApp.save_public_key ( self,
                                 private_key )
```

Zapisuje publiczny klucz RSA do pliku PEM.

Parameters

private_key Klucz prywatny, z którego generowany jest klucz publiczny.

Returns

Ścieżka do zapisanego pliku publicznego klucza.

Dodatkowo w programie istnieją 2 funkcje pomocnicze `get_drive_letters()` służąca do pozyskania nazw dostępnych dysków oraz `log_message()` do wypisania komunikatu w GUI

```
◆ get_drive_letters()
main.USBKeyApp.get_drive_letters ( self )
```

Zwraca listę aktualnie podłączonych dysków.

Returns

Zbiór nazw dysków (np. {'C:', 'D:'}).

```
◆ log_message()
main.USBKeyApp.log_message ( self,
                             message )
```

Loguje wiadomość do okna oraz do konsoli.

Parameters

message Wiadomość do wyświetlenia.


2) a.py

Drugi program służy do podpisania pliku PDF za pomocą klucza prywatnego znajdującego się na nośniku pendrive. Podobnie jak main.py, składa się on z jednej klasy SignerApp posiadającej konieczne metody

a.SignerApp Class Reference Public Member Functions | Public Attributes | List of all members

Aplikacja GUI do podpisywania plików PDF za pomocą klucza RSA. [More...](#)

Inheritance diagram for a.SignerApp:



```
graph TD
    TkTk[Tk.Tk] --> aSignerApp[a.SignerApp]
```

Public Member Functions

- `__init__(self)`
Konstruktor klasy SignerApp.
- `log_message(self, msg)`
Loguje wiadomość do okna oraz do konsoli.
- `wait_for_pendrive_with_key(self)`
Czeka na podłączenie pendrive'a z zaszyfrowanym kluczem prywatnym.
- `decrypt_private_key(self, encrypted_data, pin)`
Odszyfrowuje klucz prywatny RSA z użyciem PIN-u użytkownika.
- `decrypt_flow(self)`
Proces odszyfrowania klucza prywatnego i podpisywania pliku PDF.
- `prompt_for_pdf(self)`
Otwiera okno dialogowe do wyboru pliku PDF do podpisania.

Public Attributes

- `log` = scrolledtext.ScrolledText(self, wrap=tk.WORD)
- `drive` = drive
- `decrypt_flow`
- `private_key` = serialization.load_pem_private_key(decrypted_key, password=None)
- `prompt_for_pdf`

Program zaczyna od próby uzyskania dostępu do nośnika pendrive za pomocą metody `wait_for_pendrive_with_key()`

• `wait_for_pendrive_with_key()`

`a.SignerApp.wait_for_pendrive_with_key (self)`

Czeka na podłączenie pendrive'a z zaszyfrowanym kluczem prywatnym.

Następnie uzyskiwany i odszyfrowywany zostaje klucz prywatny używając metod `decrypt_flow()` oraz `decrypt_private_key()`

• `decrypt_flow()`

`a.SignerApp.decrypt_flow (self)`

Proces odszyfrowania klucza prywatnego i podpisywania pliku PDF.

• `decrypt_private_key()`

`a.SignerApp.decrypt_private_key (self, encrypted_data, pin)`

Odszyfrowuje klucz prywatny RSA z użyciem PIN-u użytkownika.

Parameters

- `encrypted_data` Szyfrowane dane klucza prywatnego.
- `pin` 4-cyfrowy PIN wprowadzony przez użytkownika.

Returns

Odszyfrowane bajty klucza prywatnego.

Na koniec wołana jest metoda pozwalająca użytkownikowi na wybór pliku PDF do podpisania `prompt_for_pdf()`

• `prompt_for_pdf()`

`a.SignerApp.prompt_for_pdf (self)`

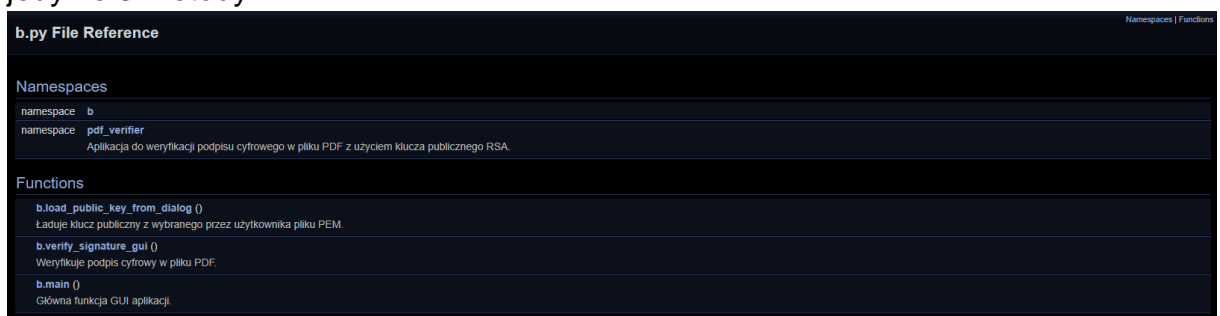
Otwiera okno dialogowe do wyboru pliku PDF do podpisania.

Tak jak w poprzednim pliku, w tym także znajduje się metoda pomocnicza do wyświetlenia użytkownikowi komunikatu `log_message()`



3) b.py

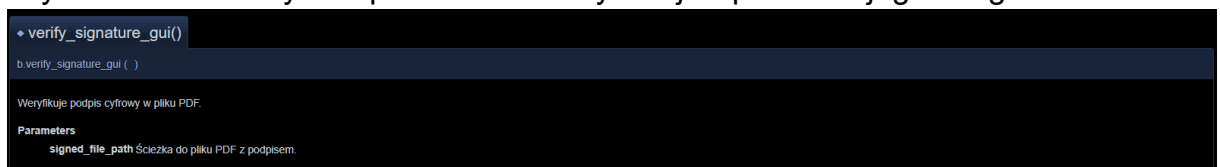
Ostatnim plikiem projektowym jest ten, odpowiedzialny za stwierdzenie wiarygodności oraz integralności pliku PDF. Nie posiada on żadnej klasy, a jedynie 3 metody.



Pierwszą metodą jest `main()` która odpowiada za uruchomienie GUI aplikacji



Następnie wykonywana jest metoda `verify_signature_gui()` dzięki której użytkownik może wybrać plik PDF do weryfikacji i sprawdzić jego integralność



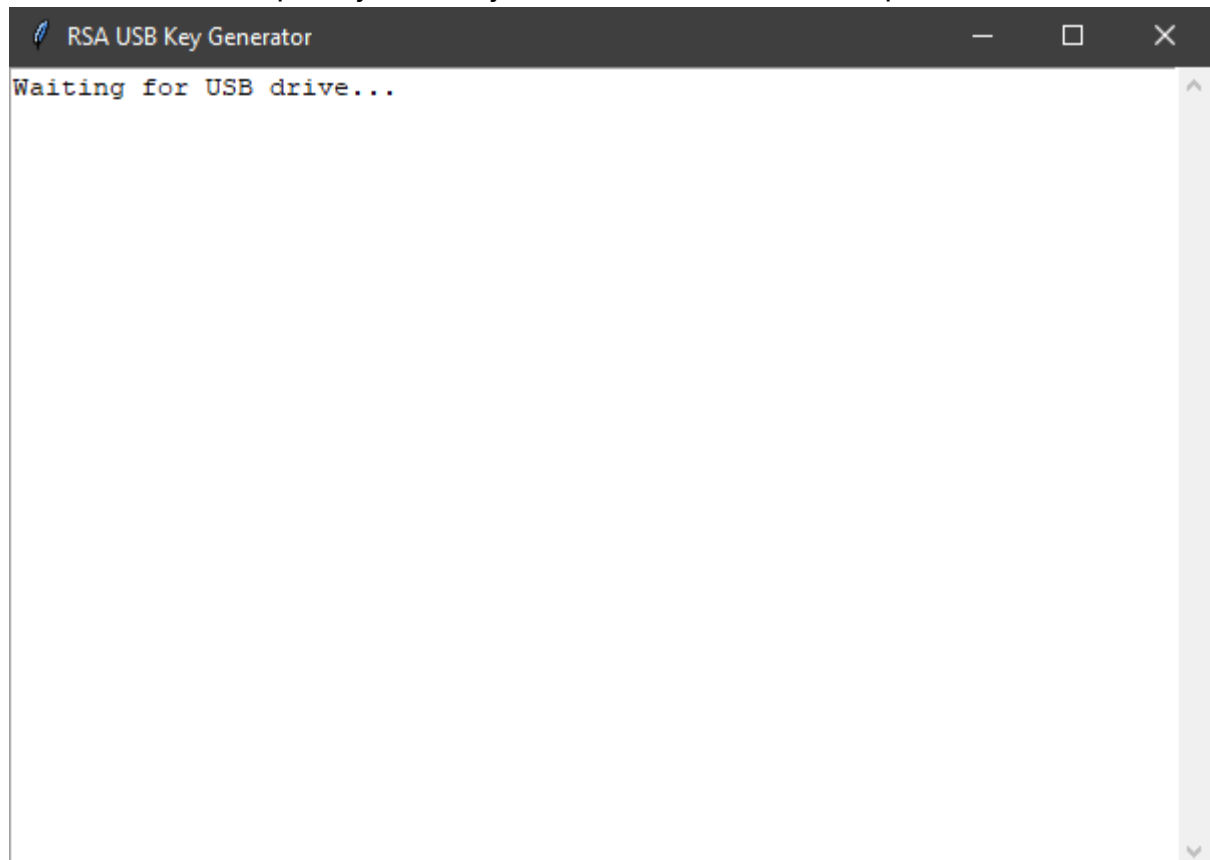
W czasie trwania metody `verify_signature_gui()` wołana jest funkcja `load_public_key_from_dialog()`, która otwiera okno dialogowe służące do wybrania klucza publicznego



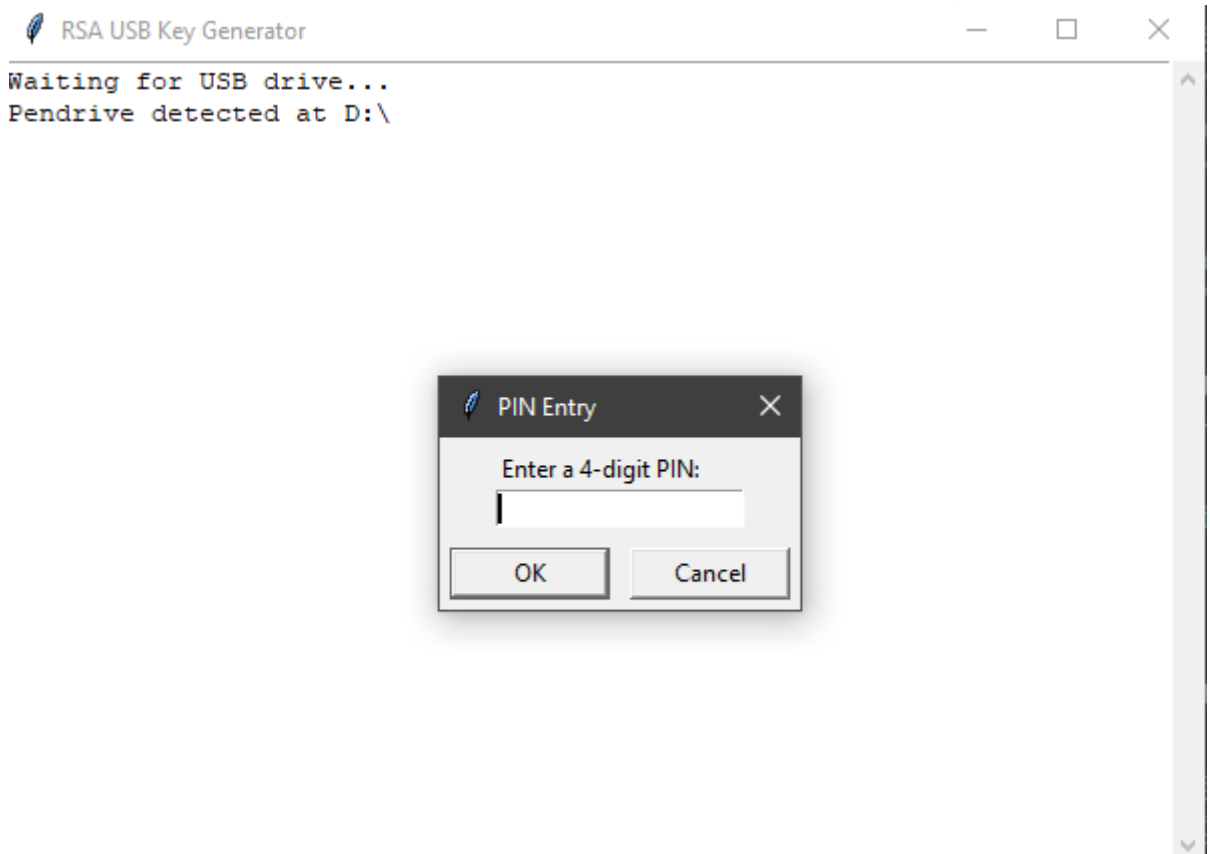
1.3 Description

Tak jak opisano wcześniej, aby uzyskać docelową funkcjonalność, trzeba przejść przez cykl 3 aplikacji. Pierwsza z nich odpowiedzialna jest za wygenerowanie klucza publicznego oraz prywatnego.

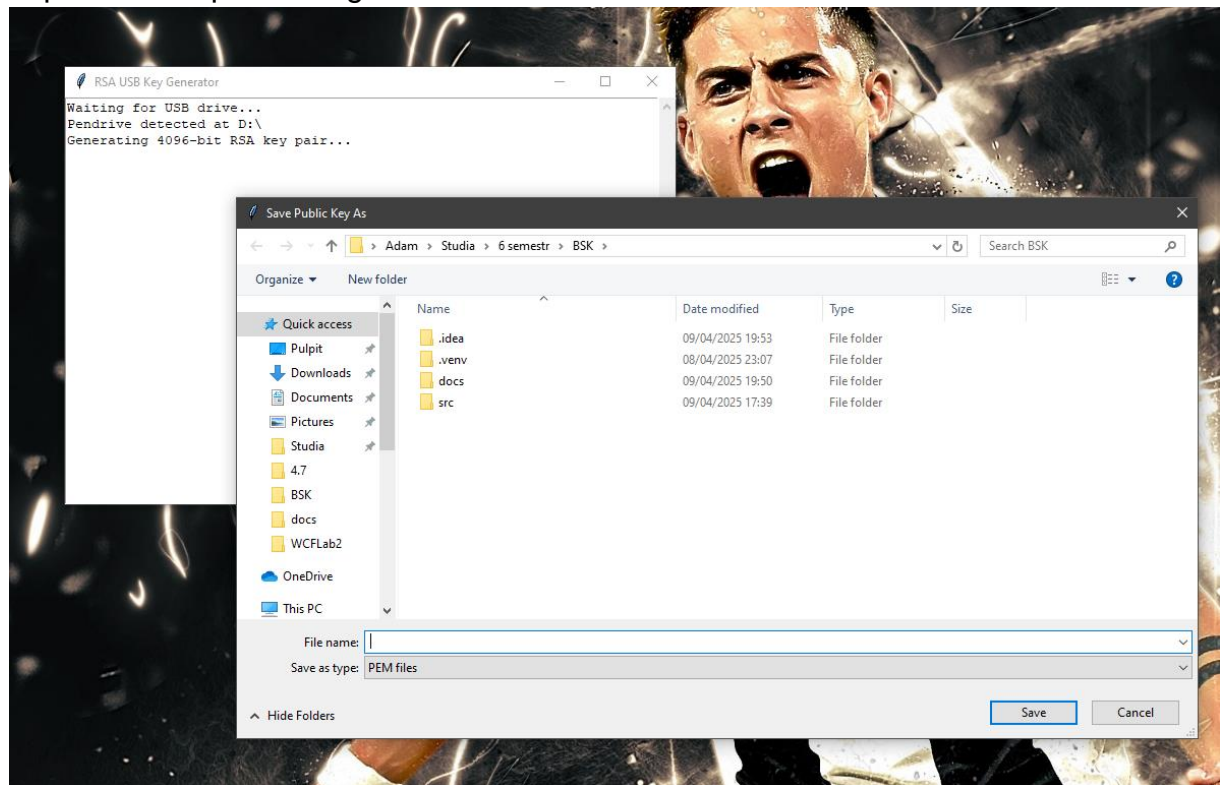
Po uruchomieniu aplikacji, oczekuje ona na włożenie nośnika pendrive



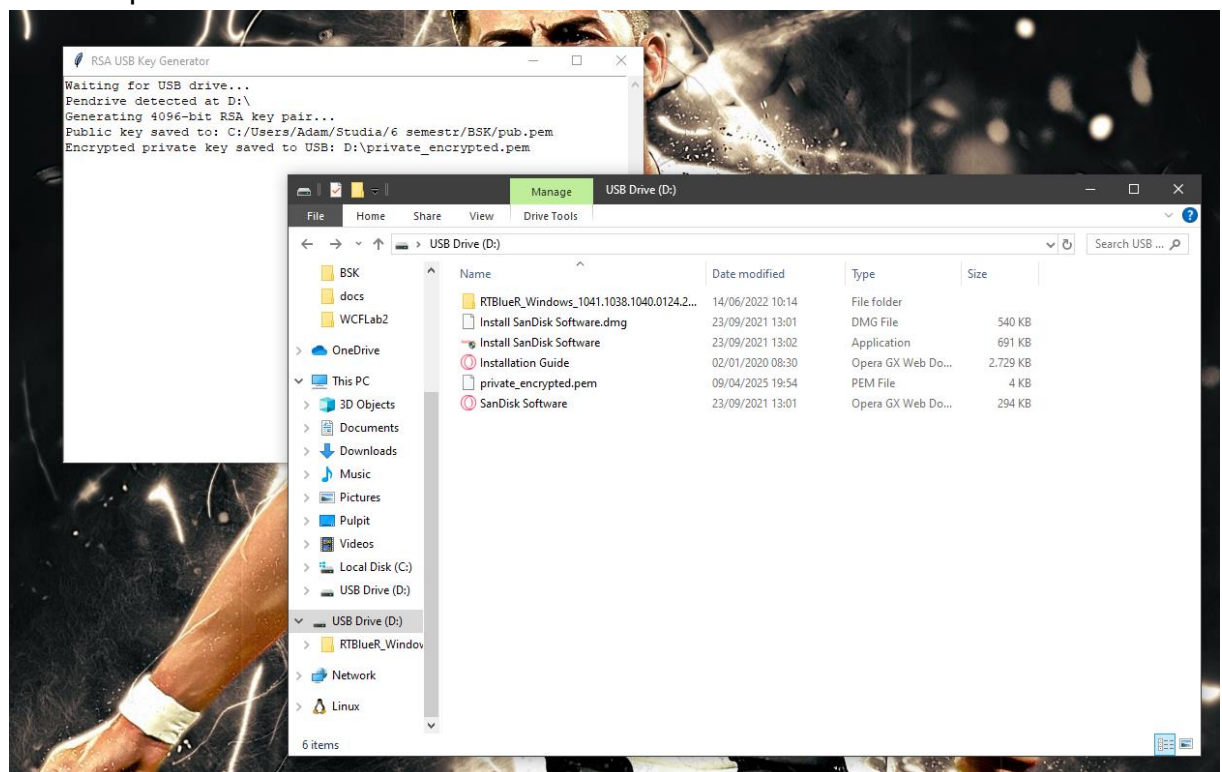
Następnie użytkownik proszony jest o wybranie 4 cyfrowego kodu PIN.



Po wybraniu kodu pin użytkownik proszony jest o wybranie nazwy oraz miejsca zapisu klucza publicznego

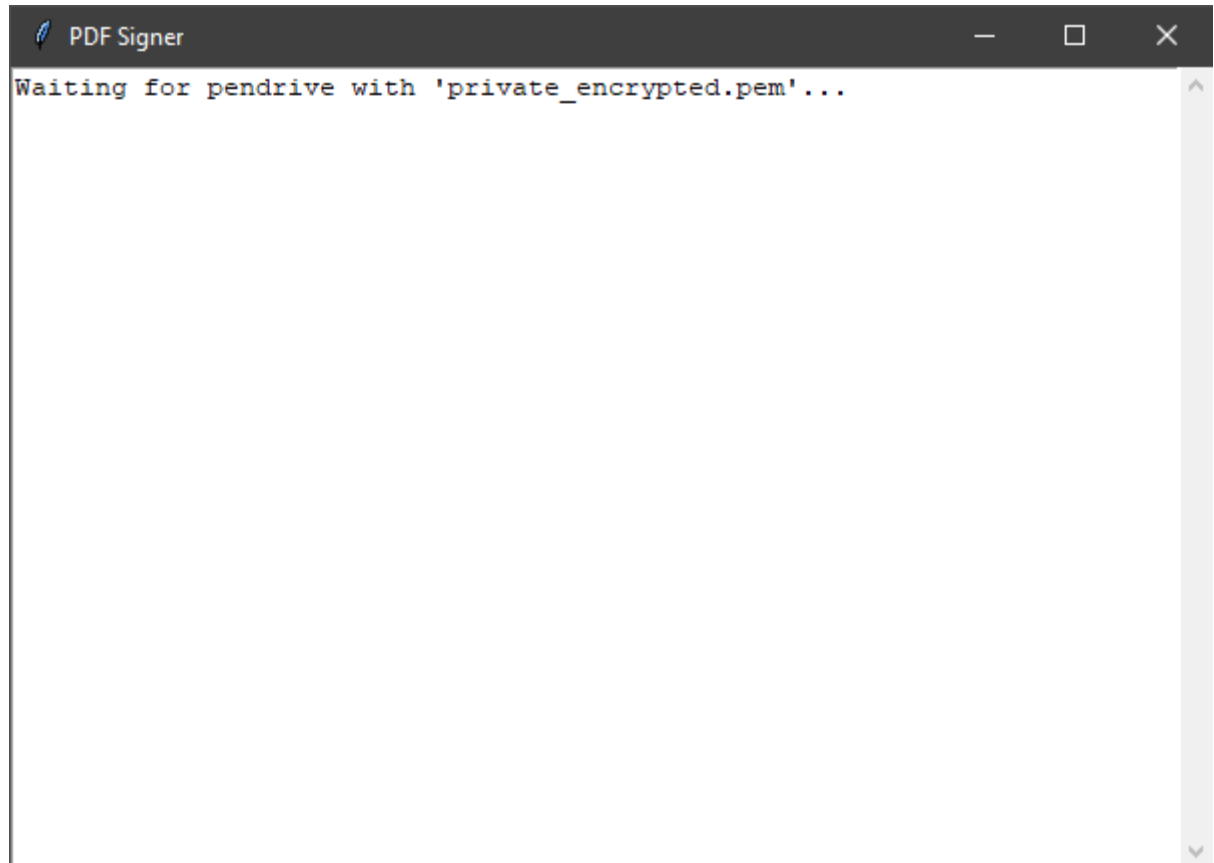


Na koniec klucz publiczny zostaje zapisany w miejscu docelowym, a prywatny na nośniku pendrive

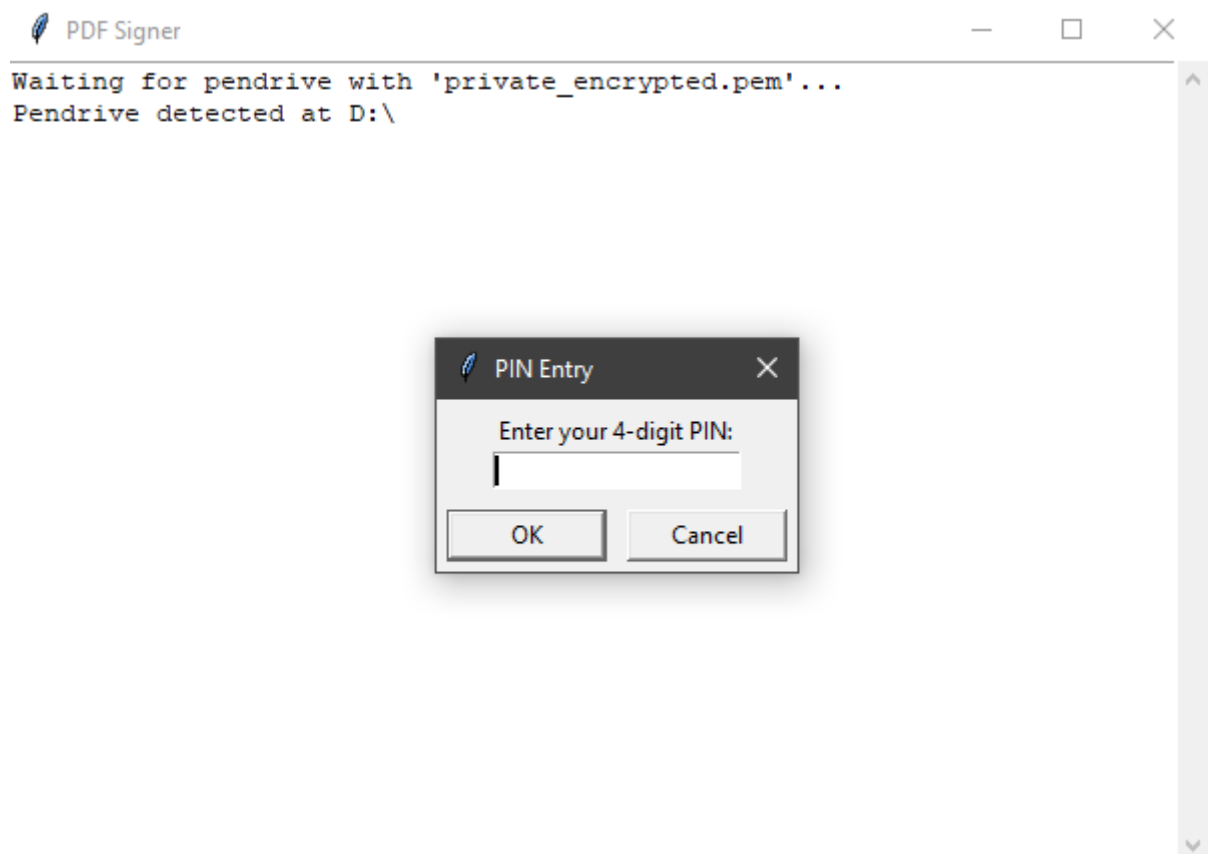


Druga aplikacja odpowiedzialna jest za podpisanie pliku PDF naszym kluczem prywatnym

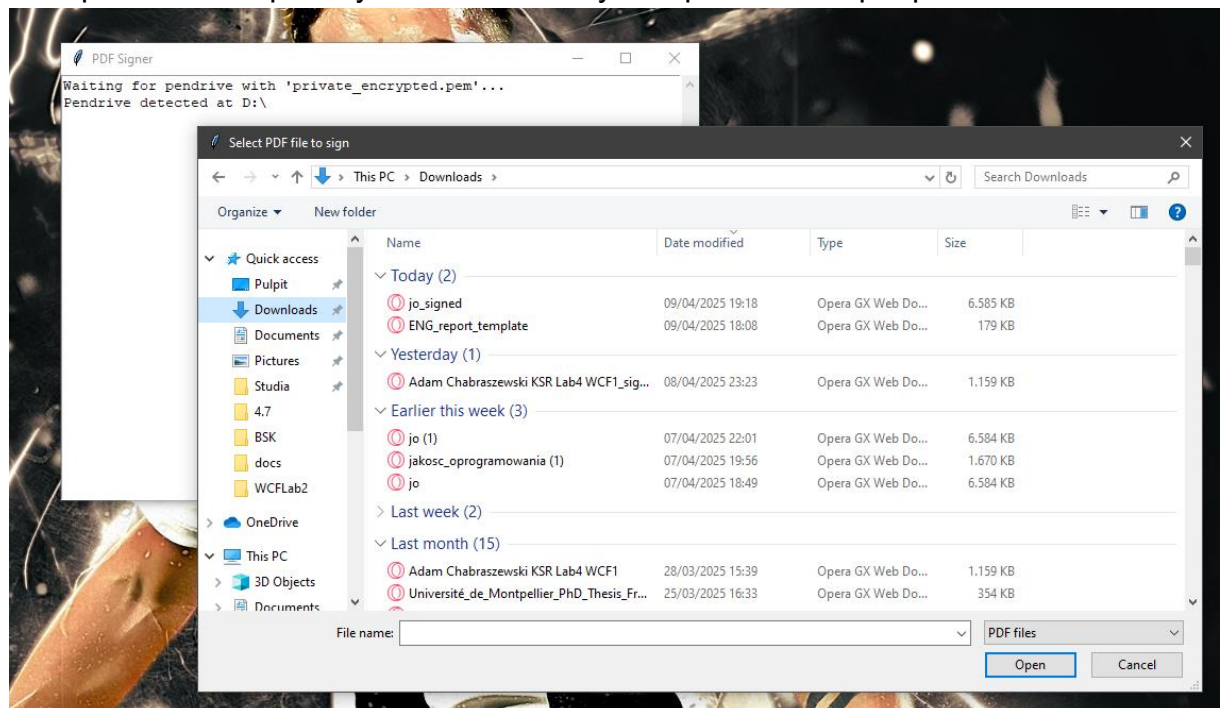
Podobnie jak w poprzedniej aplikacji, użytkownik najpierw jest proszony o włożenie nośnika pendrive



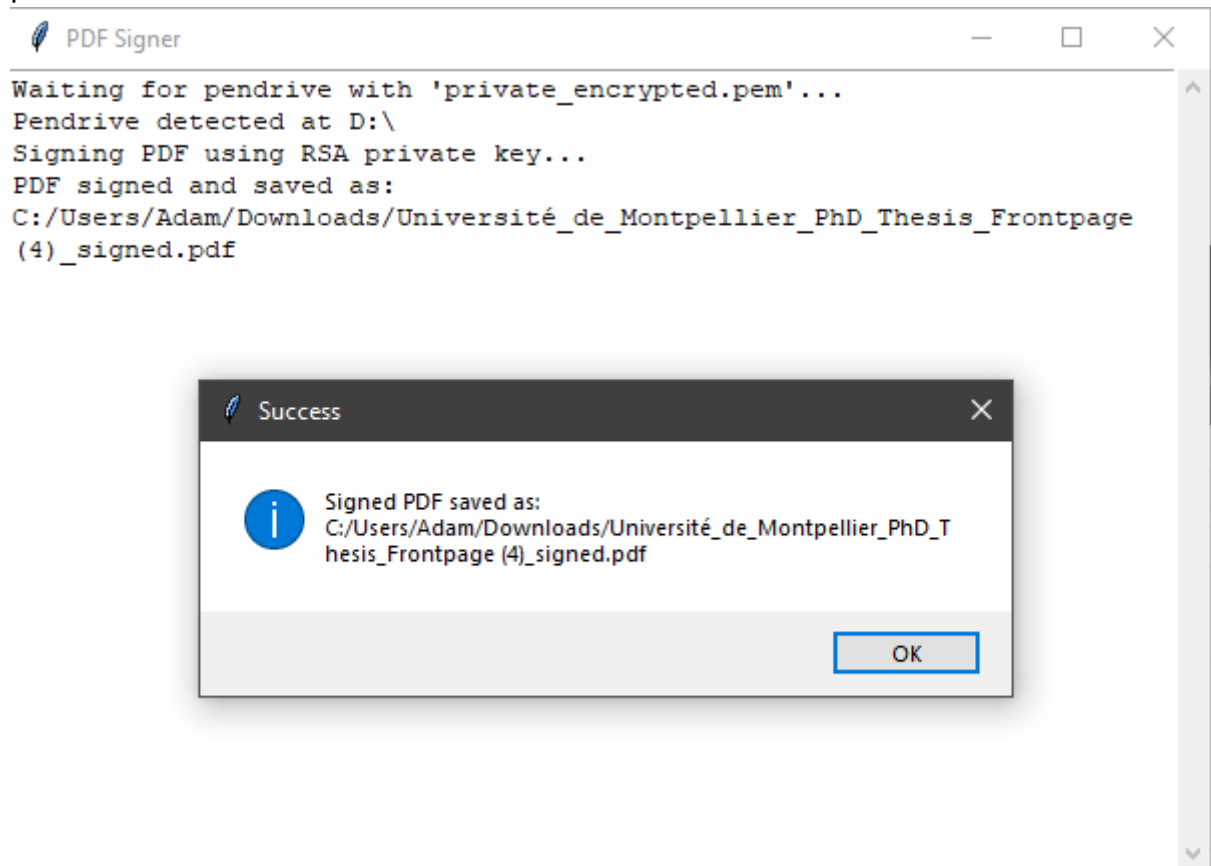
Następnie proszony jest o podanie wcześniej ustawionego kodu PIN



Po wpisaniu kodu pin użytkownik może wybrać plik PDF do podpisania

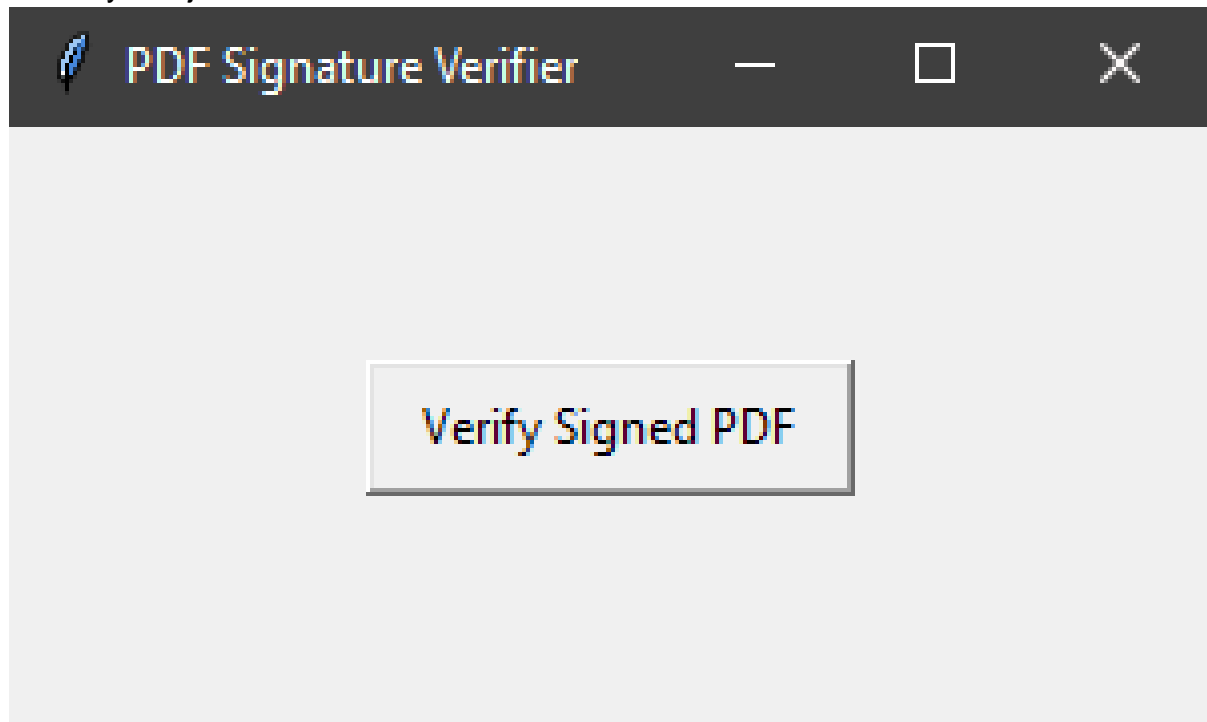


Po wybraniu pliku PDF zostaje on podpisany kluczem prywatnym z nośnika pendrive

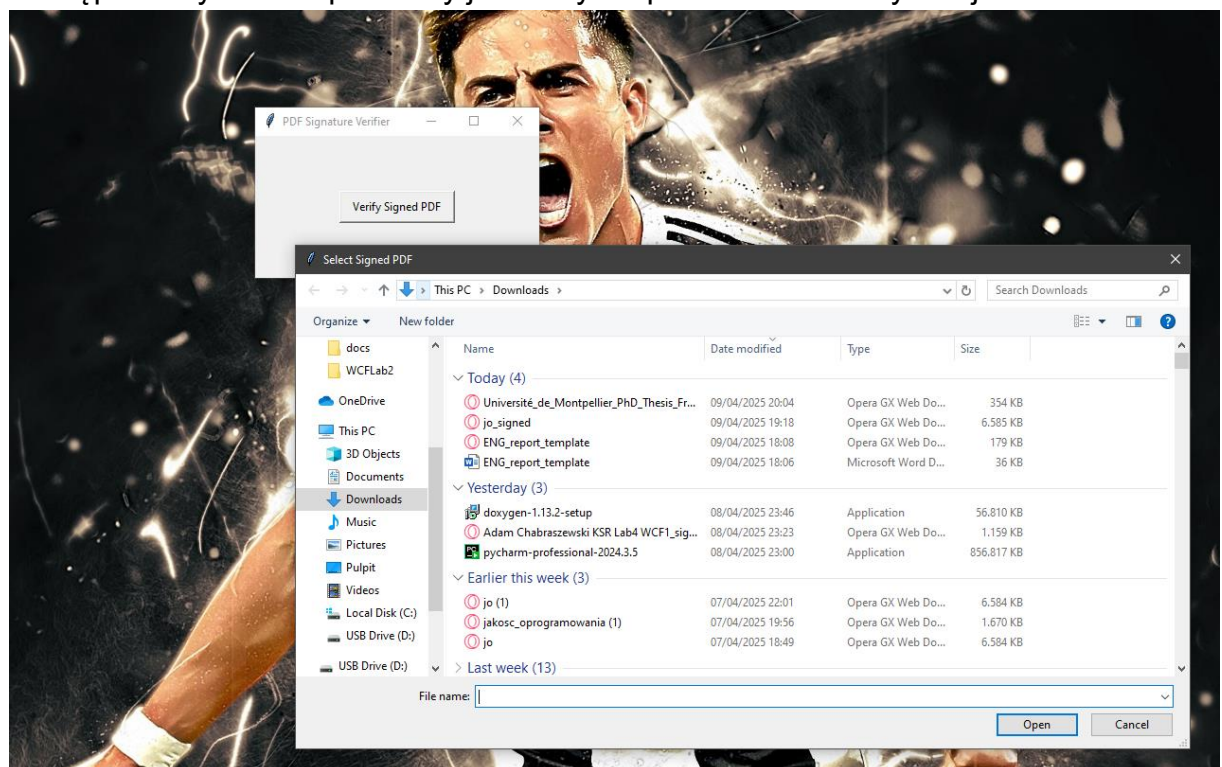


Trzecia aplikacja odpowiedzialna jest za sprawdzanie integralności pliku PDF

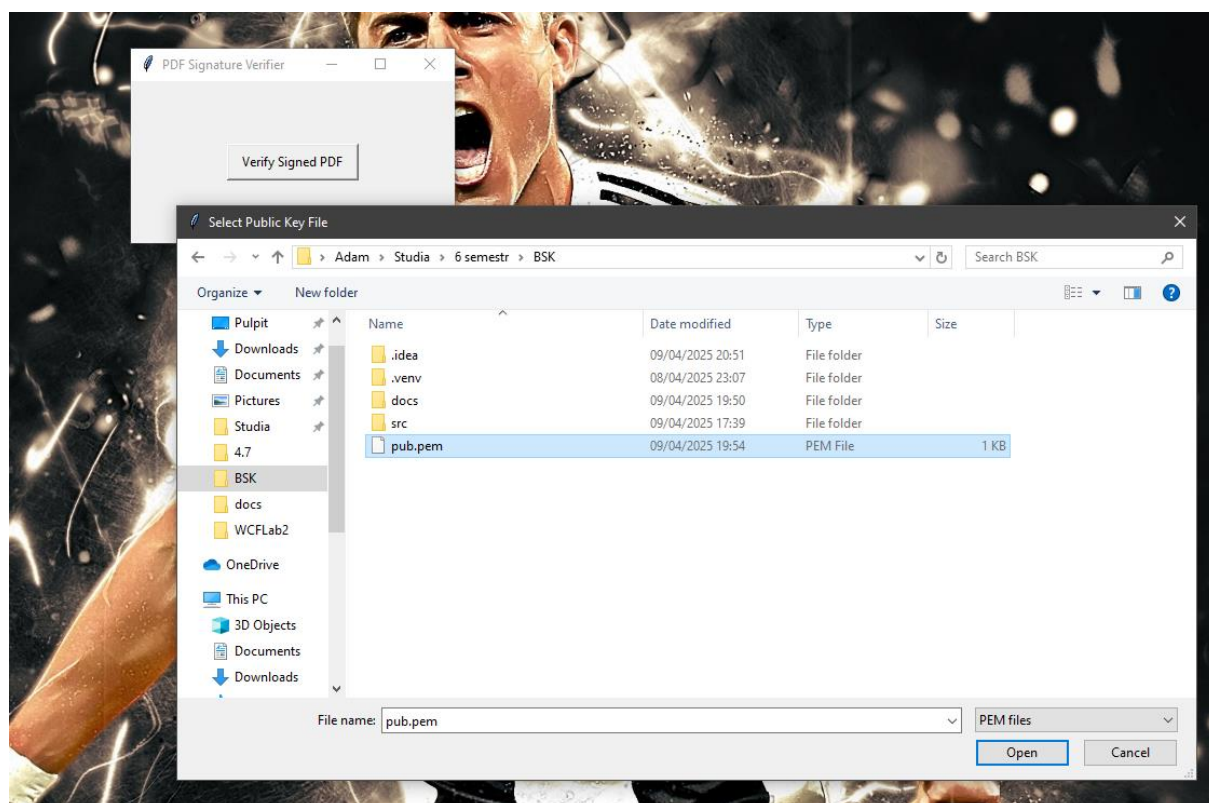
W pierwszej kolejności, aplikacja uruchamia się z przyciskiem wyboru pliku PDF do weryfikacji



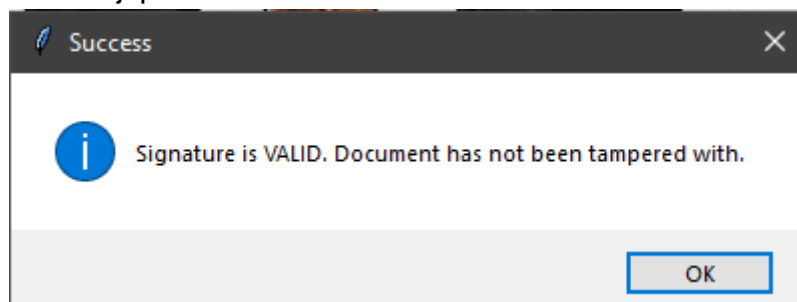
Następnie użytkownik proszony jest o wybór pliku PDF do weryfikacji



Na koniec użytkownik proszony jest o wybranie klucza publicznego do weryfikacji integralności pliku PDF

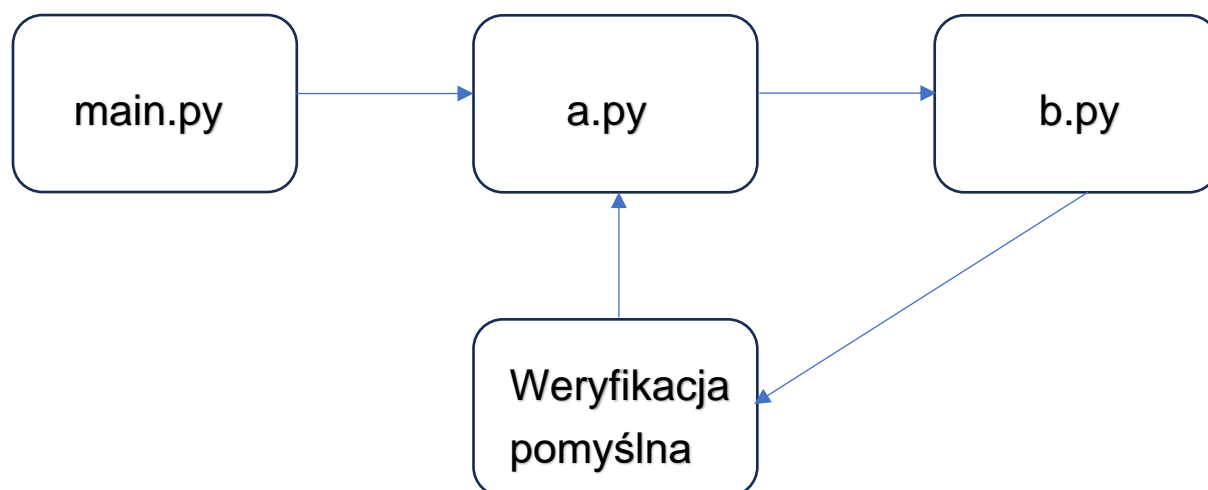


Jeżeli plik nie został zmieniony, użytkownik dostaje komunikat o pomyślnej walidacji pliku



1.4 Results

W rezultacie cykl programu prezentuje się następująco



1.5 Summary

W rezultacie, zestawienie 3 aplikacji pozwala nam uzyskać skuteczne narzędzie do kontroli integralności plików PDF. System umożliwia użytkownikowi generowanie pary kluczy kryptograficznych z użyciem kryptografii RSA, zabezpieczenie klucza prywatnego kodem PIN oraz późniejsze wykorzystanie go do podpisywania wybranych dokumentów. Integralność oraz autentyczność podpisanych plików może zostać w dowolnym momencie zweryfikowana za pomocą klucza publicznego. Całość została zaprojektowana w sposób intuicyjny i przyjazny dla użytkownika, oferując prosty graficzny interfejs oraz automatyczne wykrywanie nośnika z zapisanym kluczem.

2. Literature

- [1] [Repozytorium github](#)
- [2] [Online Doxygen documentation](#)
- [3] ...