**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. ***Opis projektu***

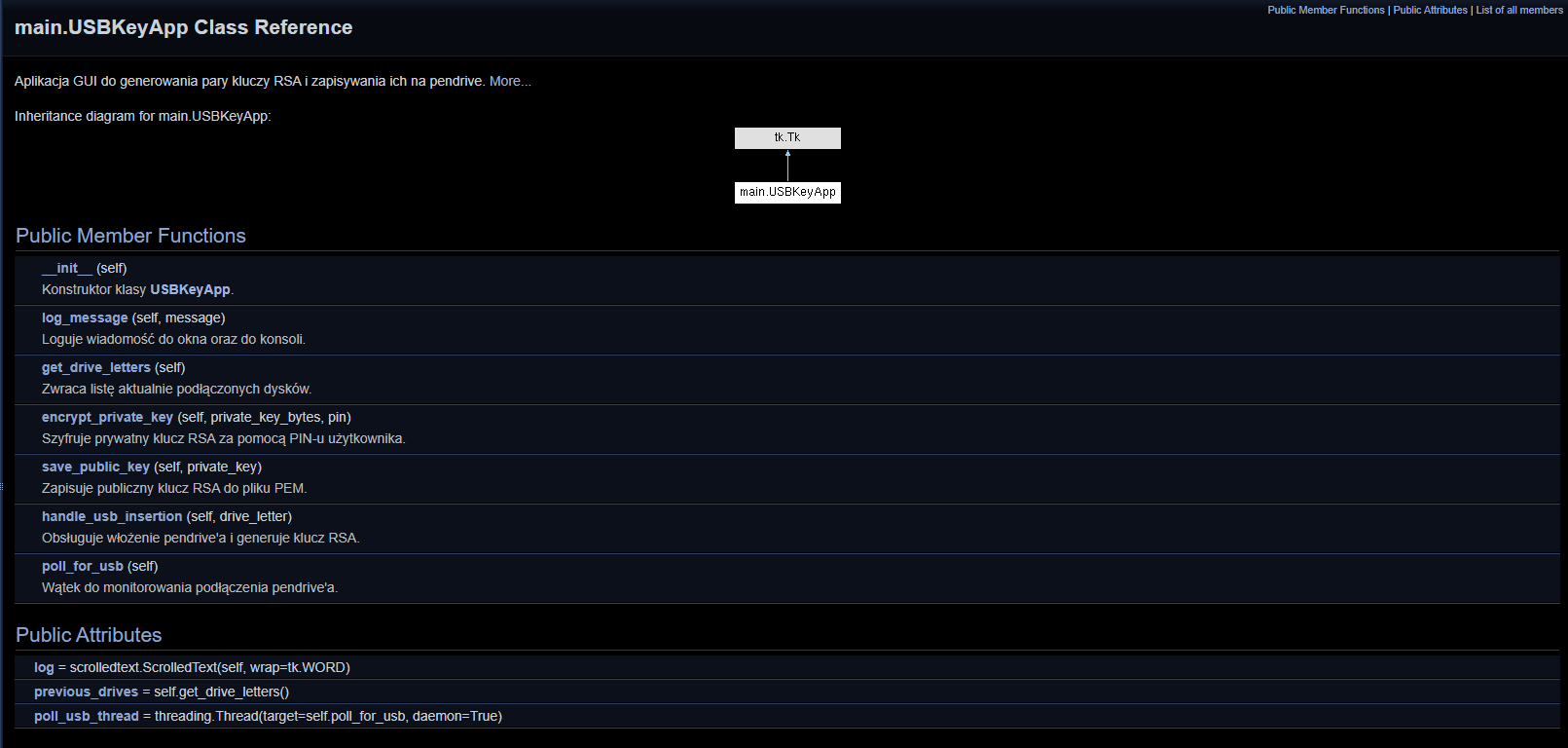
Głównym celem projektu jest stworzenie desktopowej aplikacji służącej do podpisywania i weryfikowania integralności plików PDF. Został on napisany w języku Python i jest podzielona na trzy części: pierwsza generuje klucze RSA (4096 bitów) i zapisuje zaszyfrowany klucz prywatny na pendrive, zabezpieczony 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. ***Code Description***

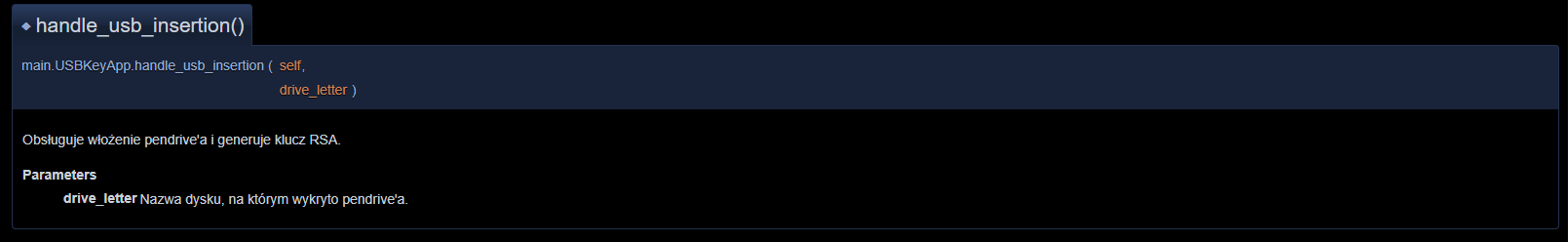
Każda z osobnych części projektu funkcjonuje jako osobna aplikacja i znajduje się w osobnym pliku

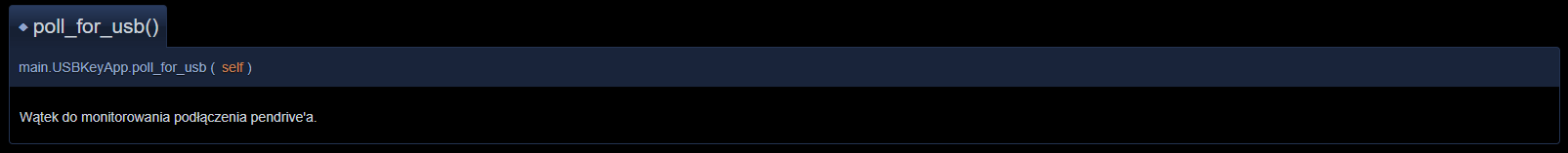
1. **main.py**

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



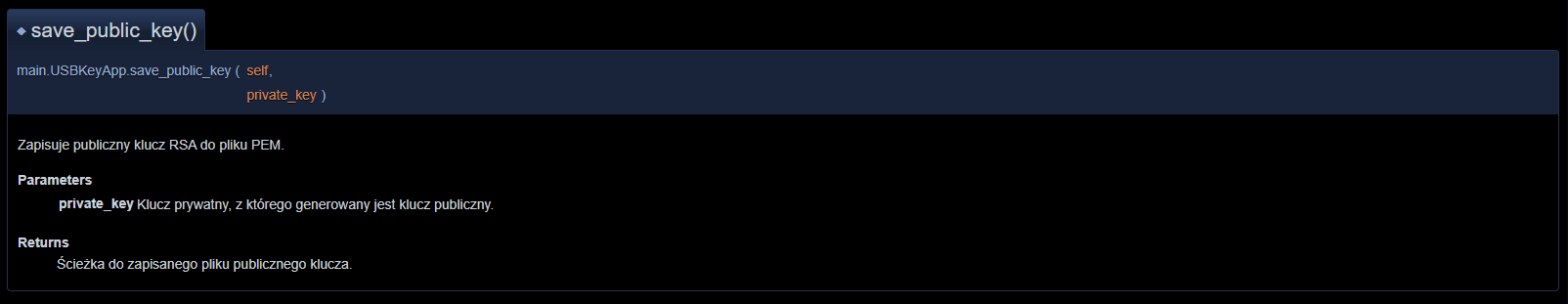
Dwie funkcje obsługujące nośnik USB to poll\_for\_usb() oraz handle\_usb\_insertion()



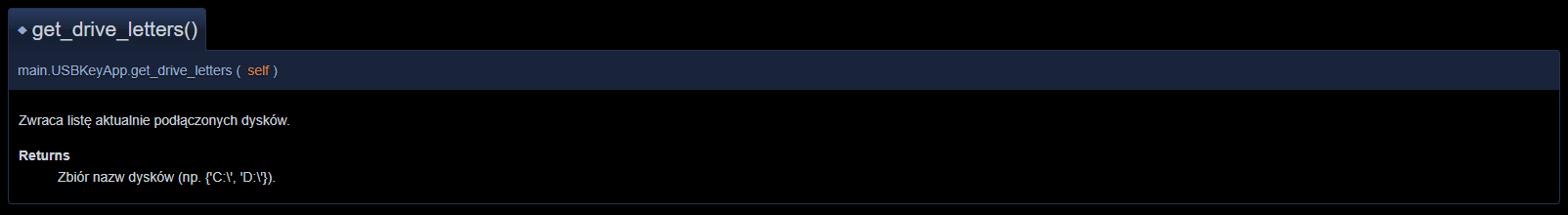


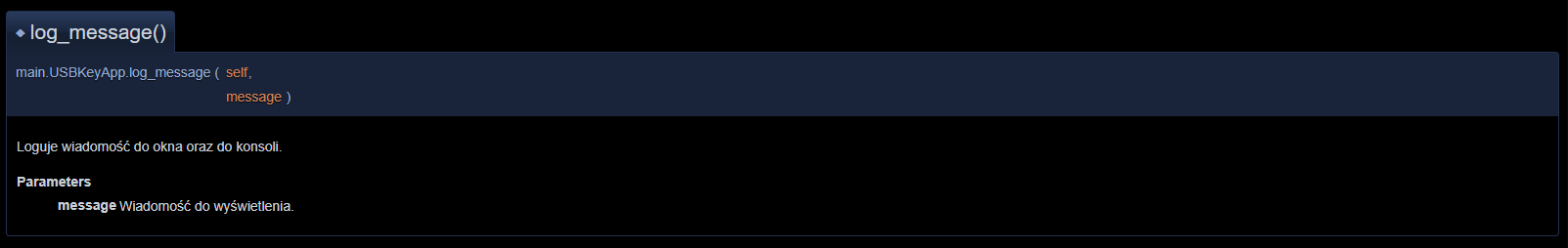
Dwie funkcje odpowiedzialne za stworzenie klucza publicznego oraz prywatnego to encrypt\_private\_key() oraz save\_public\_key()





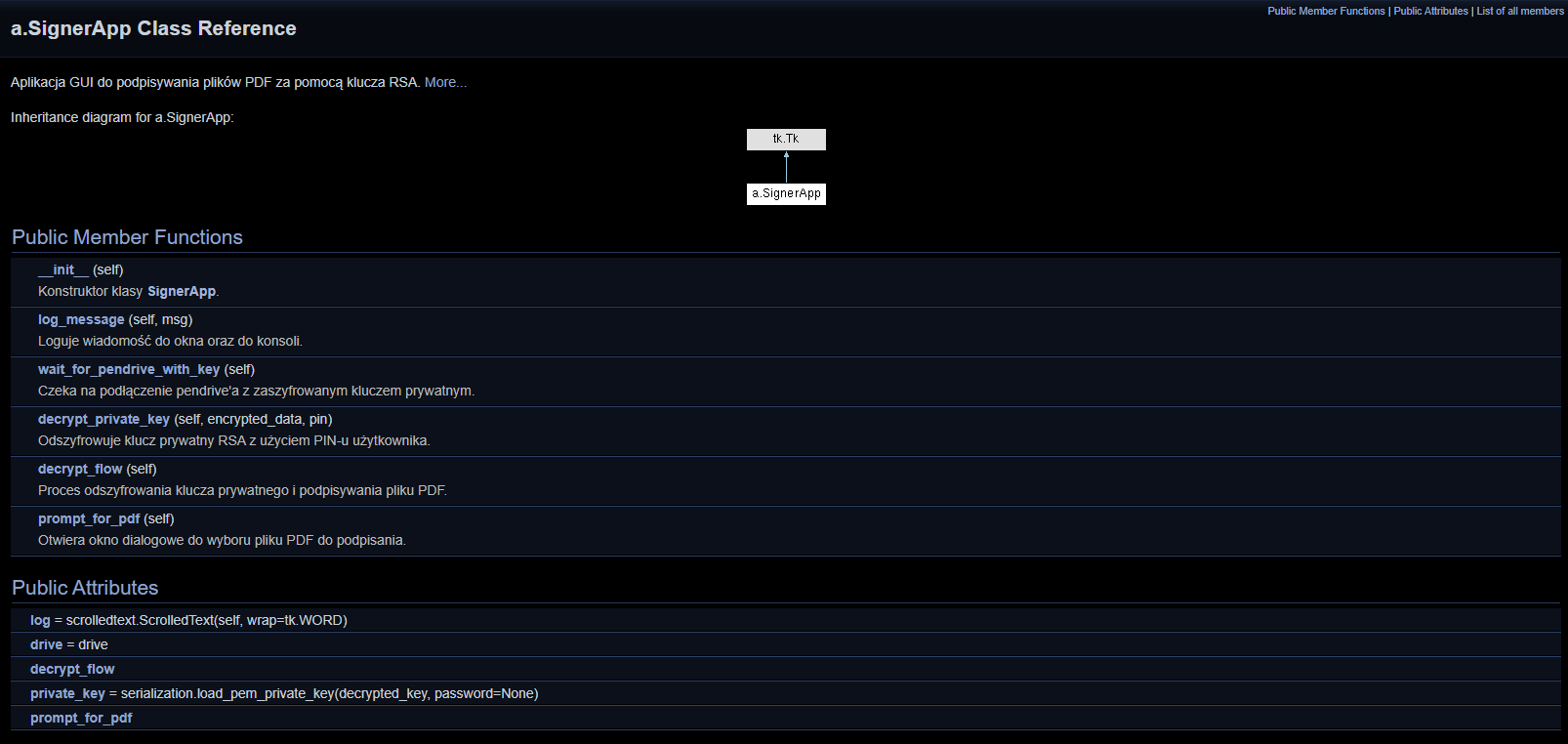
Dodatkowo w programie istnieją 2 funkcje pomocnicze get\_drive\_letters() służąca do pozyskania nazw dostępnych dysków oraz log\_message() do wypisania komunikatu w GUI



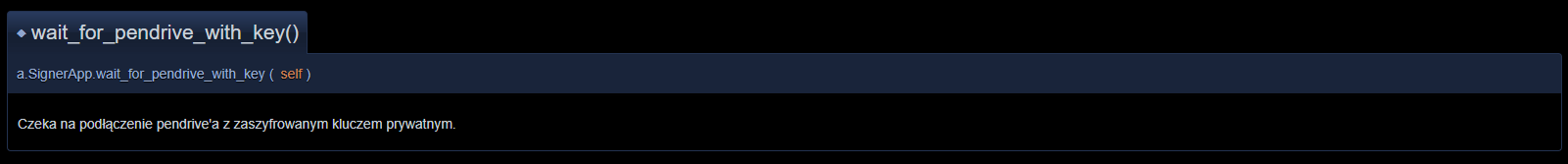


1. **a.py**

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



Program zaczyna od próby uzyskania dostępu do nośnika pendrive za pomocą metody wait\_for\_pendrive\_with\_key()



Następnie uzyskiwany i odszyfrowywany zostaje klucz prywatny używając metod decrypt\_flow() oraz decrypt\_private\_key()

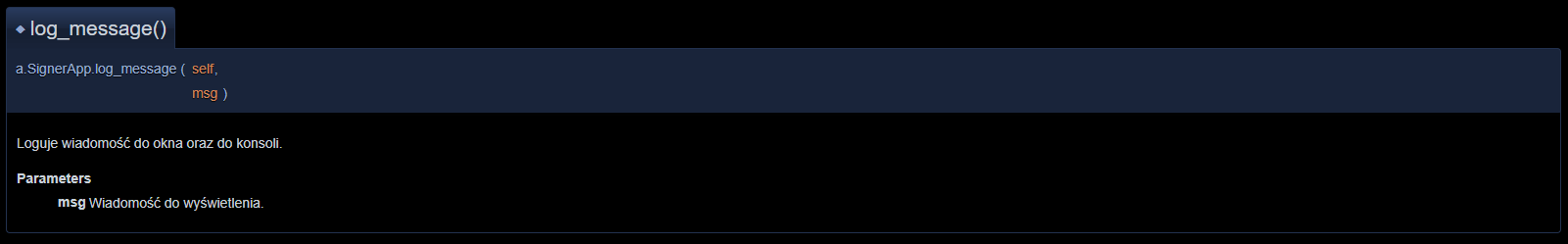




Na koniec wołana jest metoda pozwalająca użytkownikowi na wybór pliku PDF do podpisania prompt\_for\_pdf()

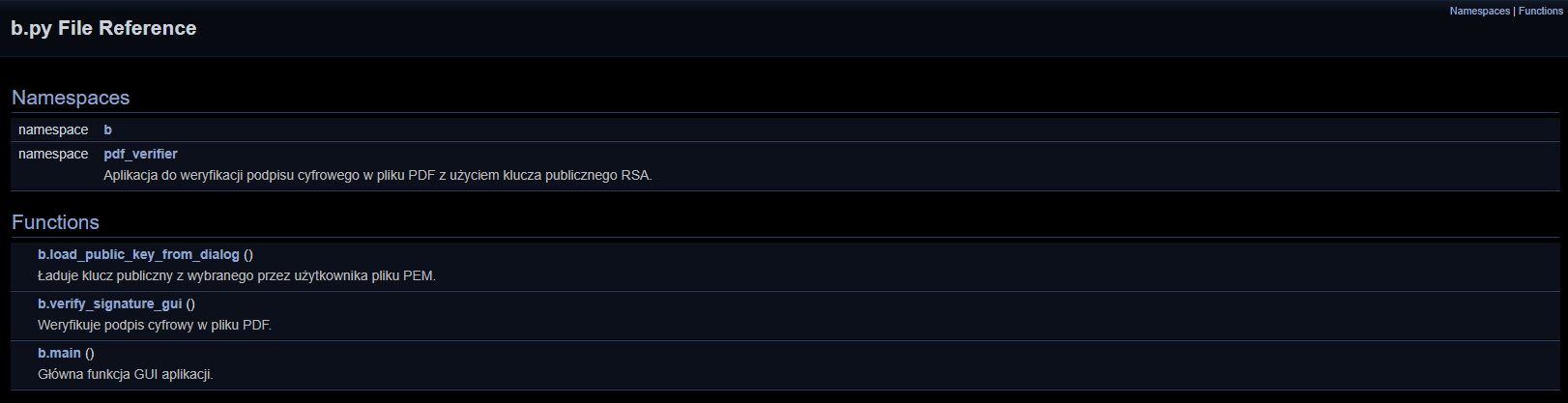


Tak jak w poprzednim pliku, w tym także znajduje się metoda pomocnicza do wyświetlenia użytkownikowi komunikatu log\_message()

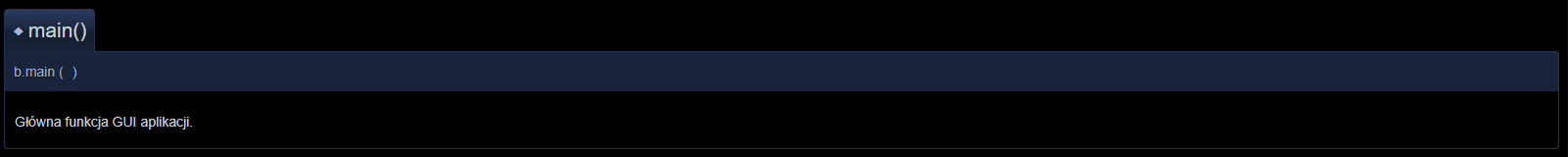


1. **b.py**

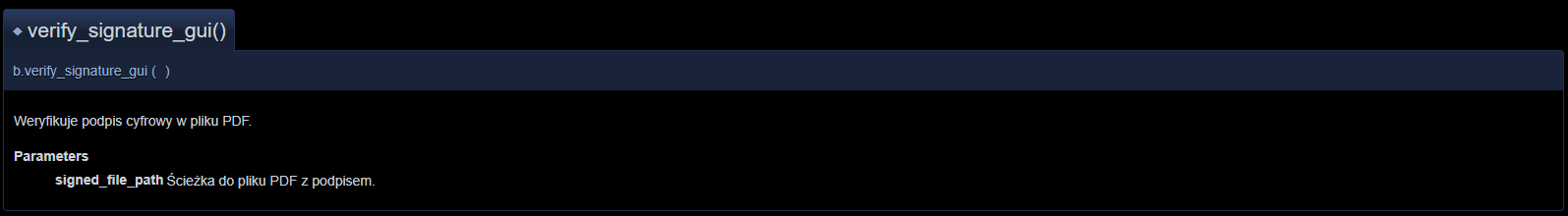
Ostatnim plikiem projektowym jest ten, odpowiedzialny za stwierdzenie wiarygodności oraz integralności pliku PDF. Nie posiada on żadnej klasy, a jedynie 3 metody.



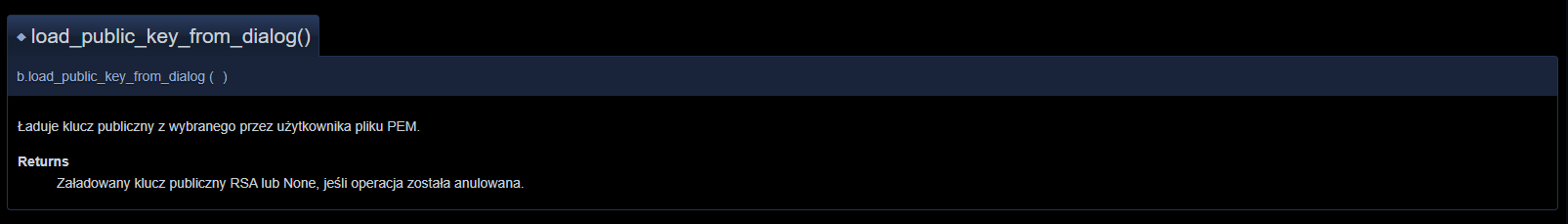
Pierwszą metodą jest main() która odpowiada za uruchomienie GUI aplikacji



Następnie wykonywana jest metoda verify\_signature\_gui() dzięki której użytkownik może wybrać plik PDF do weryfikacji i sprawdzić jego integralność



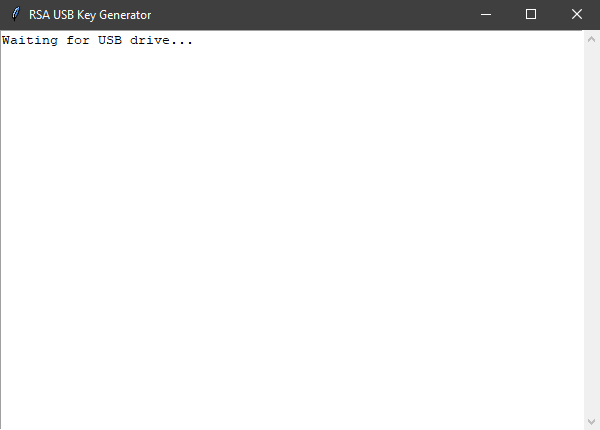
W czasie trwania metody verify\_signature\_gui() wołana jest funkcja load\_public\_key\_from\_dialog(), która otwiera okno dialogowe służące do wybrania klucza publicznego

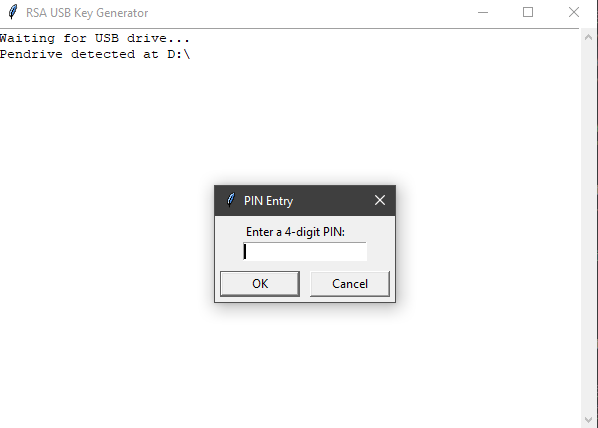


* 1. ***Description***

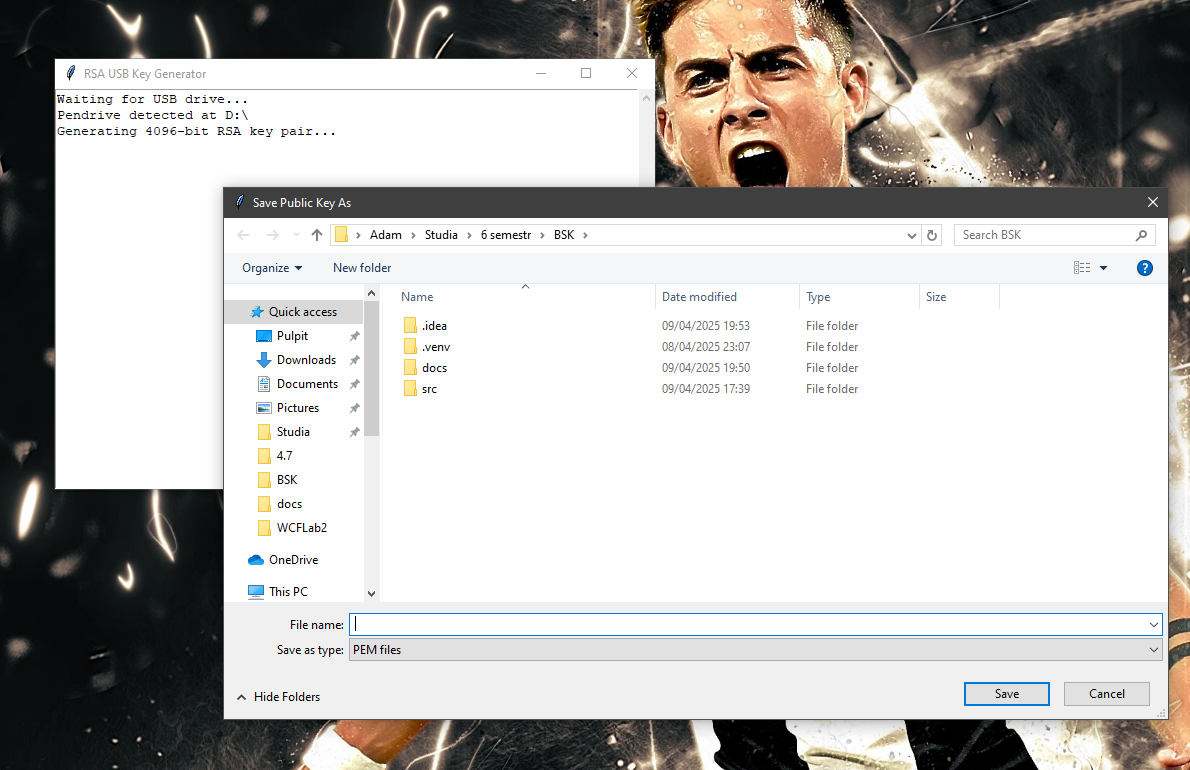
Tak jak opisano wcześniej, aby uzyskać docelową funkcjonalność, trzeba przejść przez cykl 3 aplikacji. Pierwsza z nich odpowiedzialna jest za wygenerowanie klucza publicznego oraz prywatnego.

Po uruchomieniu aplikacji, oczekuje ona na włożenie nośnika pendrive

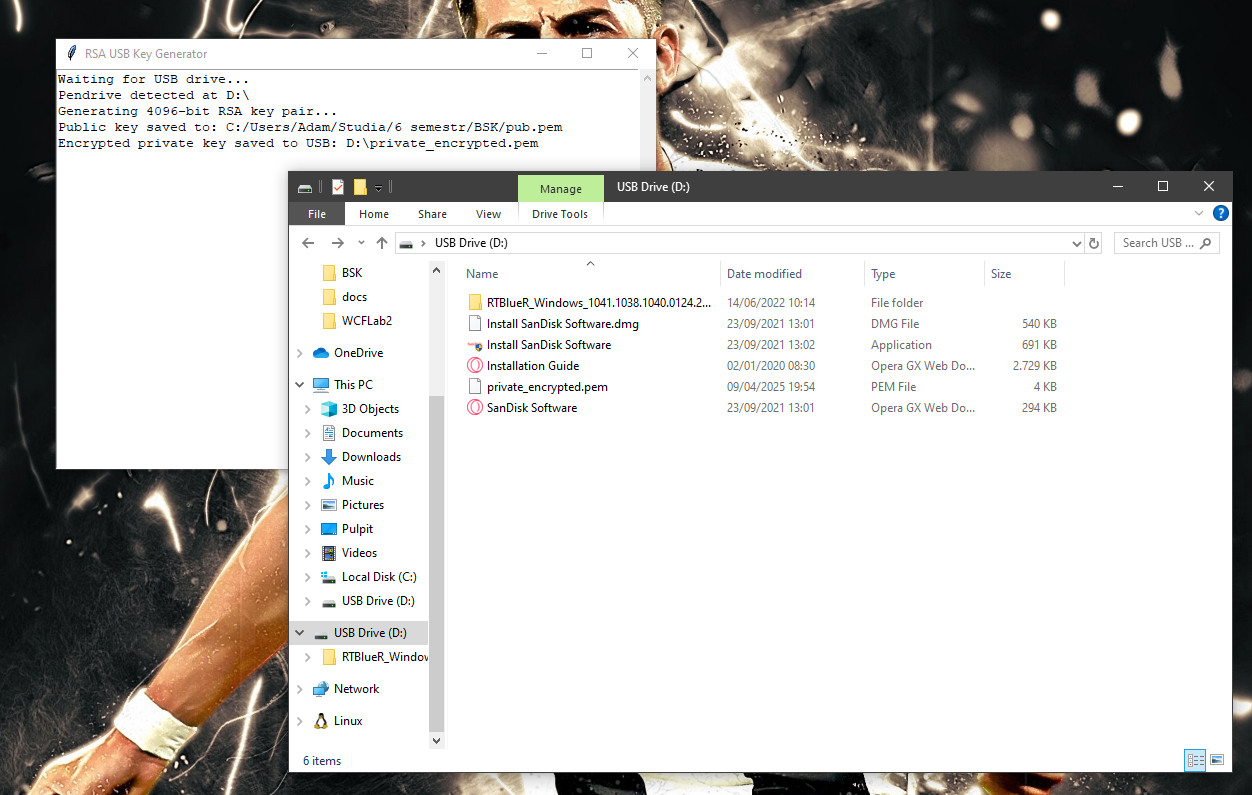


Następnie użytkownik proszony jest o wybranie 4 cyfrowego kodu PIN.

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

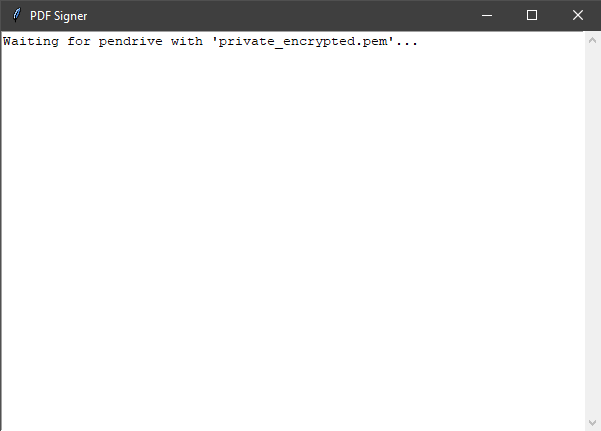


Na koniec klucz publiczny zostaje zapisany w miejscu docelowym, a prywatny na nośniku pendrive

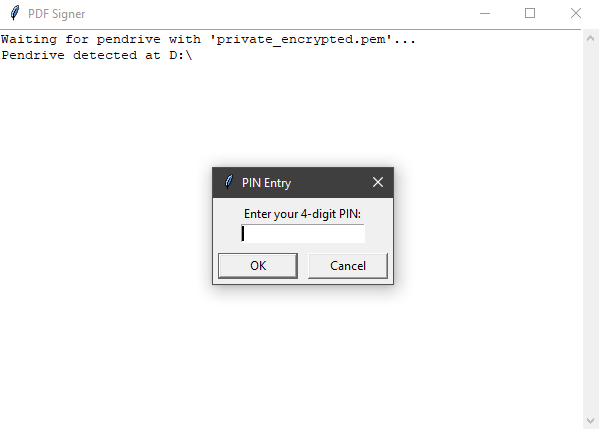


Druga aplikacja odpowiedzialna jest za podpisanie pliku PDF naszym kluczem prywatnym

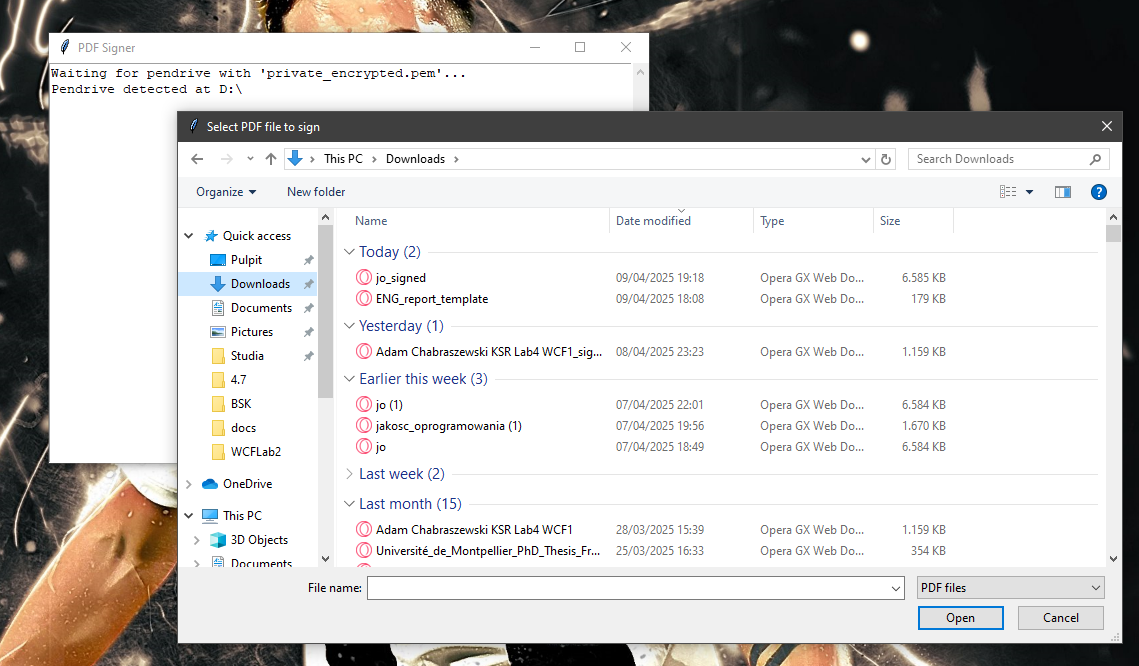
Podobnie jak w poprzedniej aplikacji, użytkownik najpierw jest proszony o włożenie nośnika pendrive



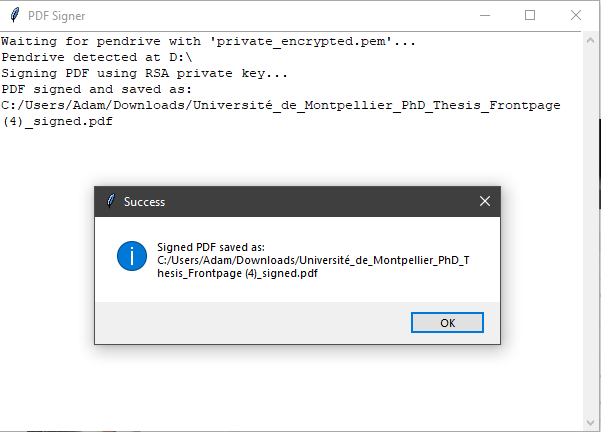
Następnie proszony jest o podanie wcześniej ustawionego kodu PIN



Po wpisaniu kodu pin użytkownik może wybrać plik PDF do podpisania

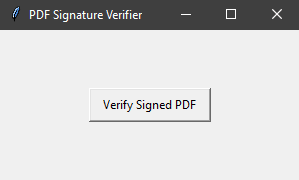


Po wybraniu pliku PDF zostaje on podpisany kluczem prywatnym z nośnika pendrive

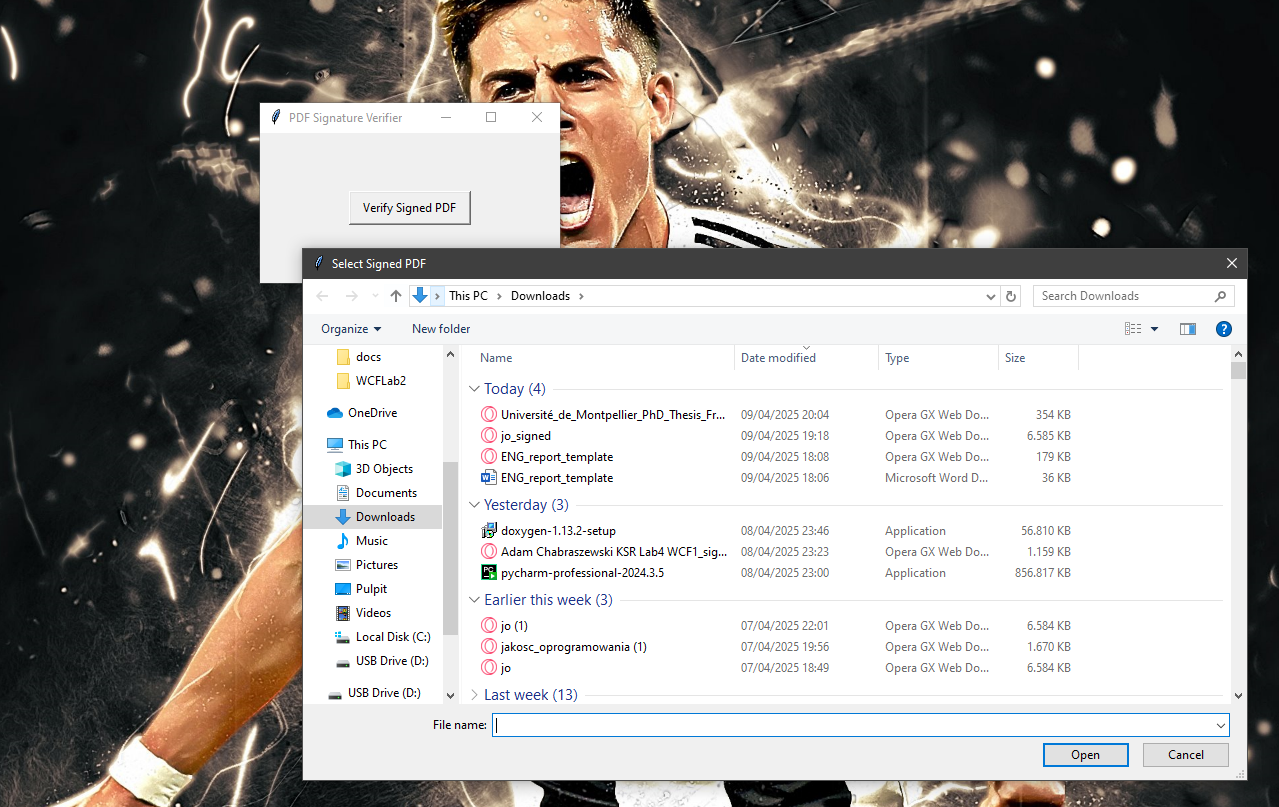


Trzecia aplikacja odpowiedzialna jest za sprawdzanie integralności pliku PDF

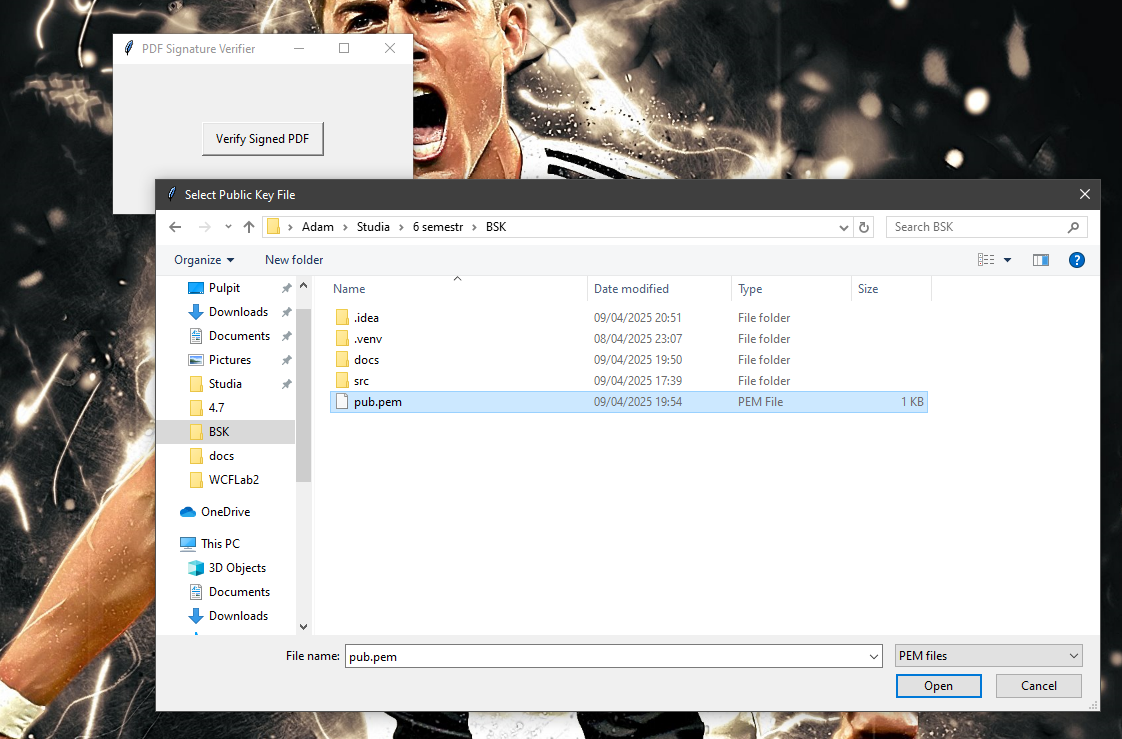
W pierwszej kolejności, aplikacja uruchamia się z przyciskiem wyboru pliku PDF do weryfikacji



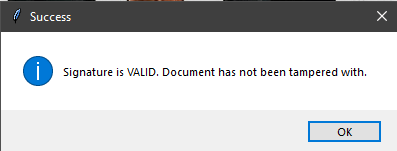
Następnie użytkownik proszony jest o wybór pliku PDF do weryfikacji



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. ***Results***

W rezultacie cykl programu prezentuje się następująco

b.py

a.py

main.py

Weryfikacja pomyślna

* 1. ***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.

1. **Literature**

[1] [Repozytorium github](https://github.com/wolekhenryk/bsk-project)

[2] [Online Doxygen documentation](https://www.doxygen.nl/manual/lists.html)

[3] …