**Security of Computer Systems**

**Project Report**

Authors:

Piotr, Kolasiński, 193275

Hubert, Szymczak, 193316

Version: 1.1

**Versions**

|  |  |  |
| --- | --- | --- |
| Version | Date | Description of changes |
| 1.0 | 09.04.2025 | Report for a control term |
| 1.1 | 12.06.2025 | Final report |

1. **Project – control term**
   1. ***Description***

The auxiliary application developed for the control term of the project enables secure generation of RSA key pairs. The application is implemented in Python with a graphical interface built using tkinter.  
The user is prompted to enter a PIN code, which is used to derive a 256-bit AES key via SHA-256 hashing. The private RSA key is then encrypted using the AES-CBC mode and saved to a selected location. The public key is saved separately in PEM format for later use in signature verification.  
The application follows the requirement of generating RSA keys with a length of 4096 bits using a secure pseudorandom generator.

* 1. ***Results***

The application consists of three main components:

* main.py – the entry point of the program
* gui.py – handles GUI creation, PIN input, and calling the key generation
* key\_generator.py – contains logic for generating, serializing, encrypting, and saving RSA keys

Upon clicking the “Generate & Save Keys” button:

* a new RSA key pair is generated,
* the private key is serialized and encrypted using AES-256 (CBC),
* the user selects where to save the encrypted private key and the public key.
  1. ***Summary***

Task 1, the auxiliary application was implemented.

* 1. ***Github repository:***

[***https://github.com/Trebuh01/secure-pdf-signer/tree/main***](https://github.com/Trebuh01/secure-pdf-signer/tree/main)

1. **Project – Final term**
   1. ***Description***

The developed solution is a secure desktop tool for digital signing and PDF verification using RSA asymmetric cryptography and USB tokens. The task was divided into three functional modules:

Key Generation and Deployment – A USB port is used to connect a device and utilized as a secure storage device to store a user's private RSA key encrypted. A 4096-bit RSA key pair is created by the application, the private key is encrypted with a user-specified 4-digit PIN, and stored securely on the USB drive. The public key is retained locally as a PEM file.

PDF Signing – A GUI utility can detect automatically the USB pen drive that holds the private key in encrypted form, ask for a PIN from the user, decrypt the private key, and use it to digitally sign a selected PDF document. The digital signature is appended to the original document, creating a \_signed.pdf file.

PDF Verification – There is a clear GUI in which the user can select an already signed PDF file and the corresponding public key file to verify the integrity and authenticity of the document. If the document has been properly signed, then it has not been tampered with.

The GUI is developed using Tkinter, and cryptographic functions are performed with Cryptodome and cryptography libraries. Decoupling utility logic, cryptographic operations, and GUI elements were achieved through modular architecture to maintain readability and upgradability.

The entire system simulates a Qualified Electronic Signature (QES) scenario that is widely used in legal and formal applications where integrity and authenticity are crucial.

* 1. ***Code Description***

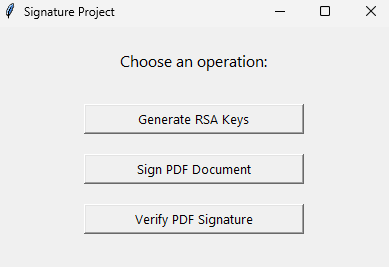
Application is organized into a number of Python modules following separation of concerns and modularity. There is one module per specific functional layer in the system, thus making codebase clean, maintainable, and extensible.  
  
**1. main.py**

This is the entry point for the application. It creates and opens the main GUI window through the SignatureInterface class. The graphical interface provides access to all the available operations: RSA key generation, PDF document signing, and signature verification.  
  
**2. gui.py**

Defines the main menu interface (SignatureInterface) through the tkinter library. It provides three main options:  
  
Generate RSA Keys  
  
Sign PDF Document  
  
Verify PDF Signature  
  
Each option launches a dedicated window that is to be used for the respective operation, invoking the related logic module.  
  
**3. key\_deployment.py**  
Installs the key generation logic, encryption, and RSA key storage in an USB drive:  
  
A 4096-bit RSA key pair is generated.  
  
Private key is encrypted using AES-256 in CFB mode with the encryption key derived from a user-input 4-digit PIN using SHA-256 hash.  
  
Private key is kept on the USB device encrypted and the public key in PEM form to a user-specified location.  
  
The module monitors events on the USB devices in a background thread and deploys upon detecting a new drive.  
  
**4.signer.py**

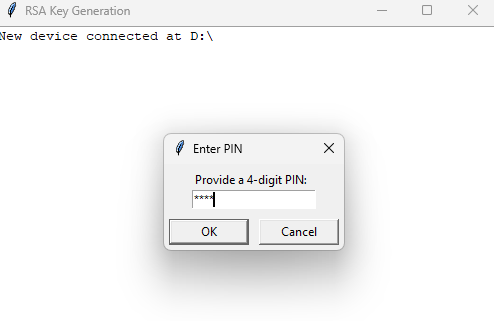
Handles digital signing of PDFs:  
  
Waits for USB device with encrypted private key file.  
  
Requests PIN and decrypts private key through the KeyDecryptor.  
  
Assists the user in selecting a PDF file and a digital signature.  
  
Signed file is saved to a new file with \_signed.pdf suffix.  
  
**5. verifier.py**  
Supports functionality for verification of authenticity and integrity of a signed PDF:  
  
The user selects a signed PDF and a public key file.  
  
Signature (the last 512 bytes of the file) is being extracted from the content.  
  
Verifies the SHA-256 hash of the original content using RSA signature verification.  
  
**6. utils.py**  
Stores reusable utility classes:  
  
USBUtility: fetches connected drives and writes binary information.  
  
RSAKeyHandler: generates and serializes RSA keys.  
  
KeySecurity: encrypts and decrypts private keys.  
  
DriveWatcher: monitors the presence of target files on attached drives.  
  
KeyDecryptor: decrypts private key data with AES-256 and imports it as an RSA object.  
  
DigitalSigner: computes SHA-256 hash of file content and signs with RSA.

* 1. ***Dzialanie kodu***

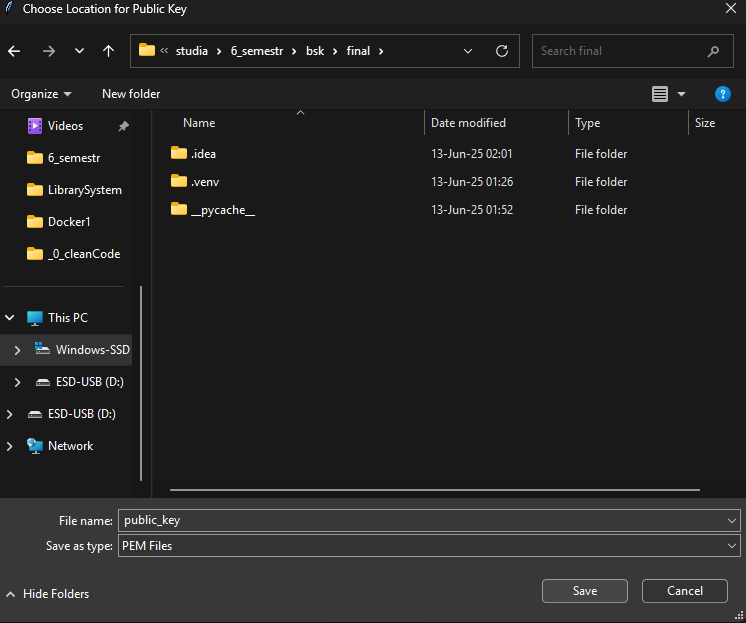


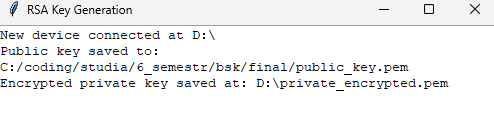
*This is the main menu of the application. It allows the user to choose one of three actions:*

* *Generate RSA keys and save them securely*
* *Sign a PDF document using a private key from a USB*
* *Verify the signature of a previously signed PDF using a public key.*

**

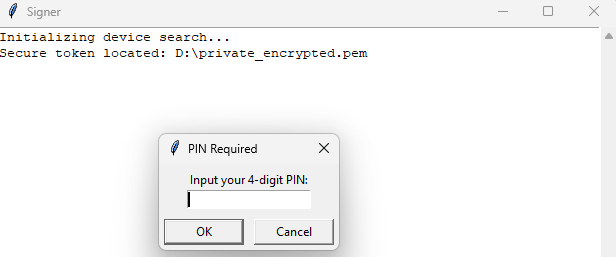
*This window appears during RSA key generation. When a USB device is detected, the user is prompted to enter a 4-digit PIN, which will be used to encrypt the private key before saving it securely to the USB drive.*

**

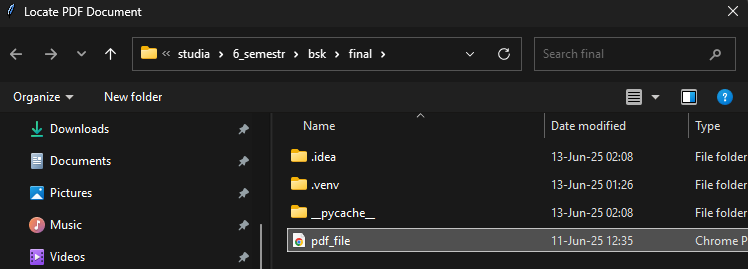
**

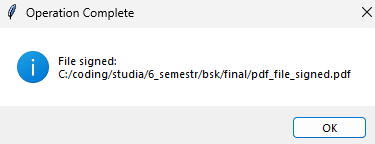
*After entering the PIN, the user is asked to choose where to save the public key in PEM format. Once confirmed, the application displays a summary:*

* *The public key is saved to the selected folder.*
* *The private key is encrypted with the provided PIN and saved securely to the USB drive.*

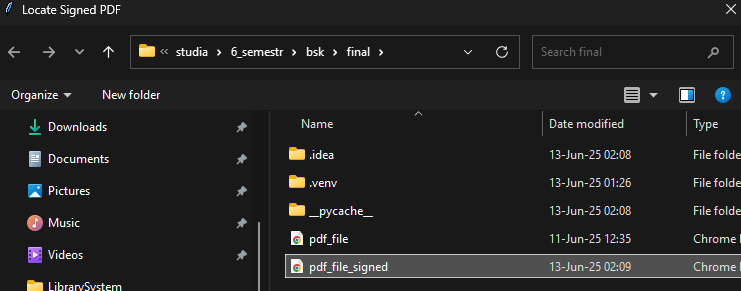
**

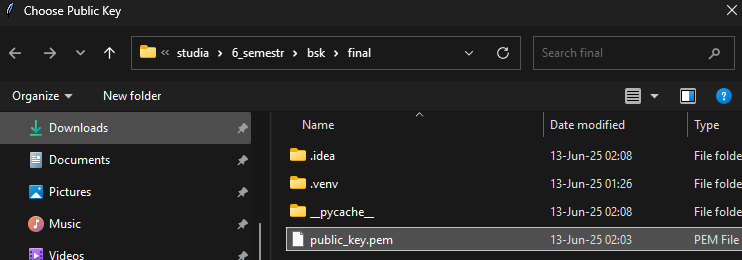
*During the signing process, the application detects the USB token containing the encrypted private key. It then prompts the user to enter the 4-digit PIN to decrypt the key and proceed with signing a PDF document.*

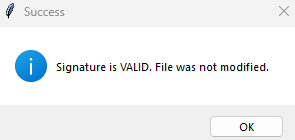




*After decrypting the private key, the user is prompted to select a PDF file for signing. Once selected, the document is digitally signed, and a new file named \*\_signed.pdf is created and saved in the same folder.*





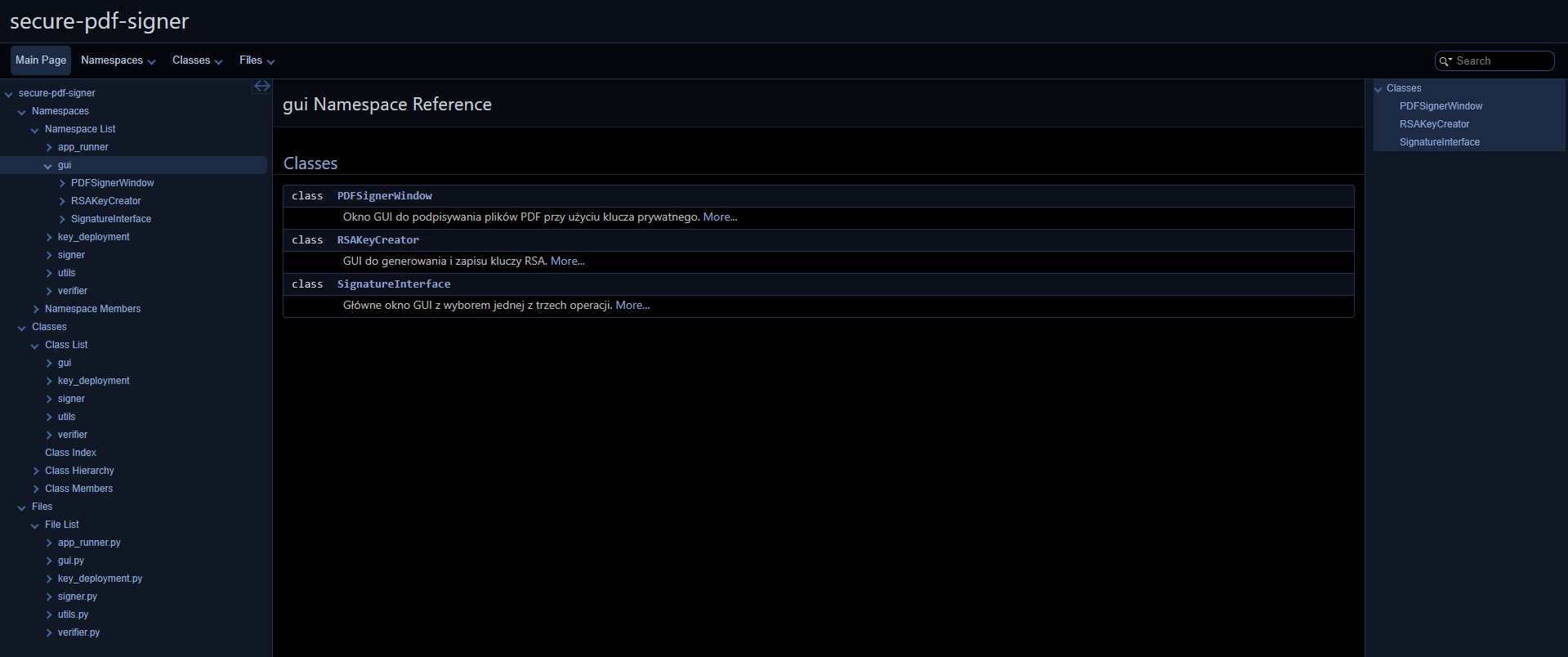


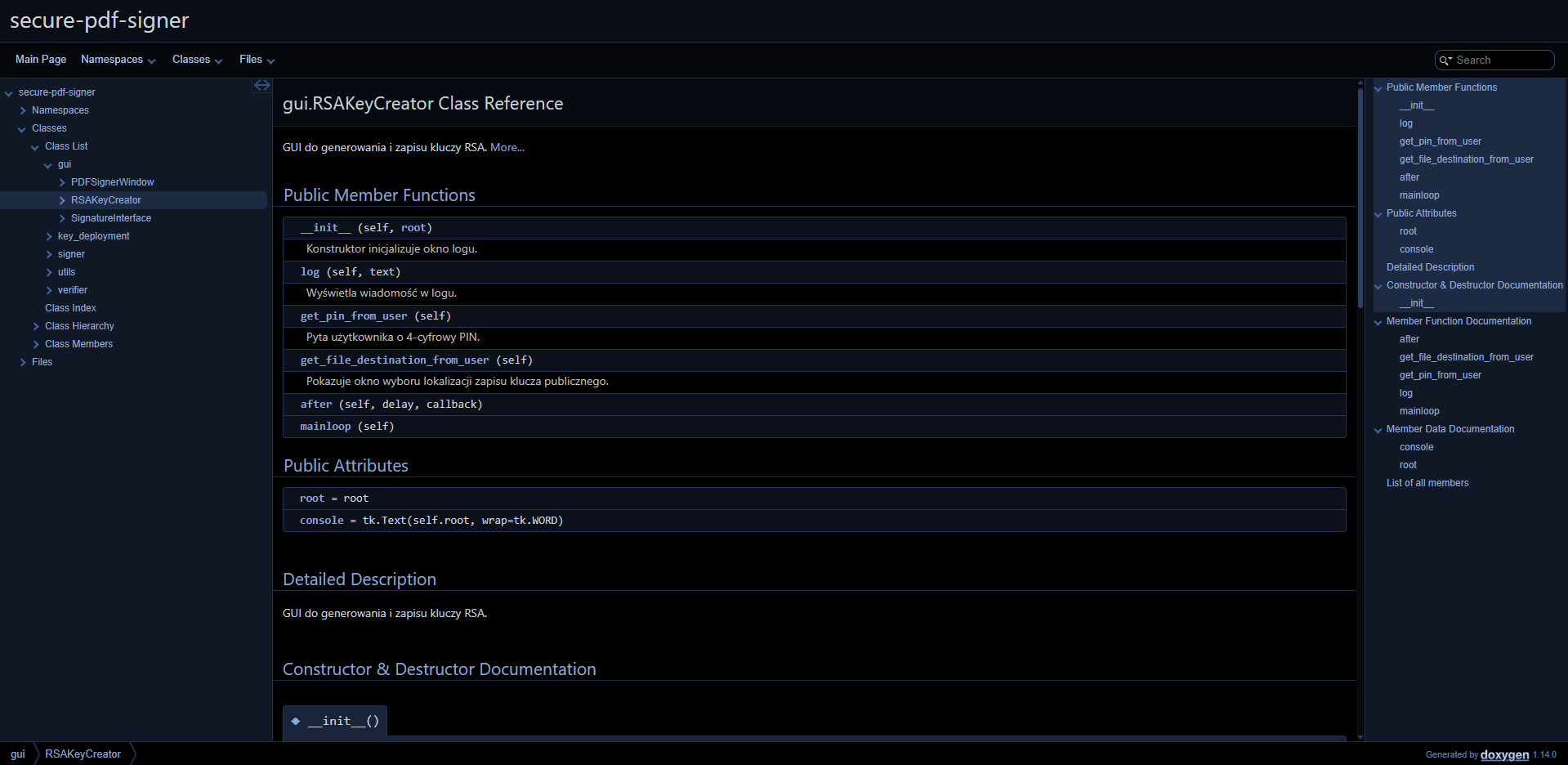
*In the verification step, the user selects a signed PDF and the corresponding public key file. The application checks the signature and confirms whether the file is authentic and unmodified.*

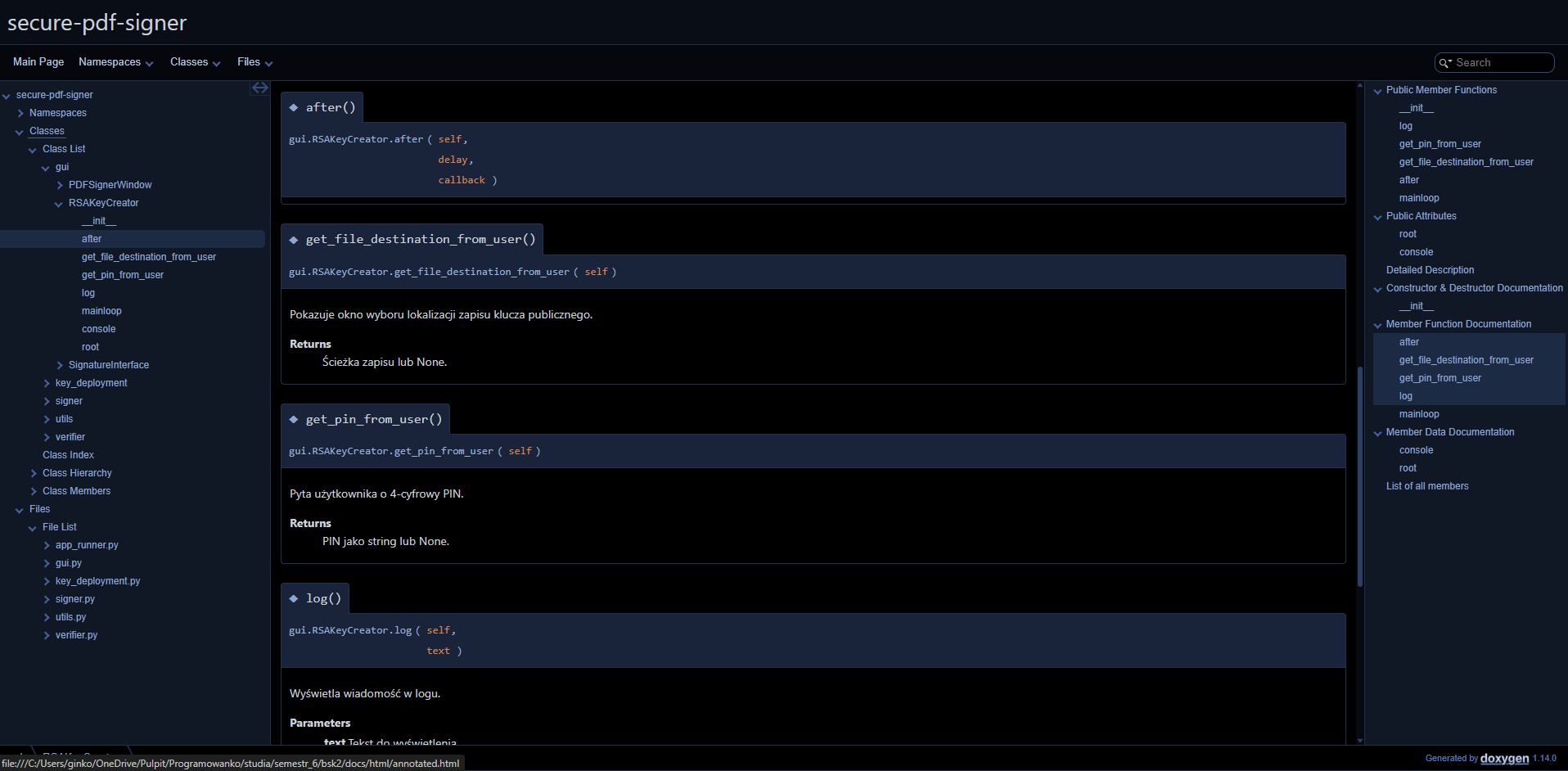
* 1. ***Summary***

The application allows users to securely generate RSA key pairs, sign PDF documents based on private keys in USB drives, and validate signed documents based on public keys. Security was ensured through AES encryption of derived private keys from a PIN key and modular code organization.

The application was developed in Python with tkinter being used for the graphical user interface and cryptographic functions with cryptography/pycryptodome. In addition to the code base, technical documentation was developed with Doxygen that gives a detailed explanation of the structure and functionality of the source code.







1. **Literature**

[1] Online Doxygen documentation, https://www.doxygen.nl/manual/lists.html