Security of Computer Systems

Project Report

Author:
Adam, Zarzycki, 193243

Version: 2.1

Versions

Version	Date	Description of changes
1.0	11.04.2025	Creation of the document
2.0	08.06.2025	Added the “Solution” part, updated the “AuxiliaryApp” part
2.1	09.06.2025	Attached Doxygen documentation

Table of contents

1. Information about the project

1.1. Project summary

The main goal of the project is to realize a software tool for emulating the qualified electronic signature, i.e. signing *.pdf documents. The goal is to fully emulate the process, including the hardware toll needed for person identification.[1]

1.2. Project requirements

PROJECT SUBMISSION – Presentation during classes		
	Task	Points
1	Generation of RSA keys, storing private key in a secure form – 2nd auxiliary application	3
2	Usage of hardware tool (pendrive with encrypted private key) during signing procedure, automatic key detection must be implemented	3
3	Generation of correct signature file – the modified *.pdf with signature details, associated with signed document	4
4	Presenting of correct and incorrect validation of signature by user B (pointing out resistance to document modification).	5
5	Presentation the functioning main and auxiliary applications during project submission	5
REPORTS – The report is evaluated only after project presentation		
6	Partial report (presentation only) for the control meeting (+ code, + presentation during classes)	5
	Minimal requirements: - Presentation: e.g. possibility of generating RSA keys (auxiliary application with GUI), basic project of main application (2 points). - Code in *GitHub* repository (3 points).	
7	Project report (+ code, pointing bibliography in the report)	15
	- Description of realised task (4 points). - Description of key application functionality, pointing out code fragments (4 points). - Code documentation using Doxygen (5 points). - pointing out the bibliography (1 points). - code in *GitHub* repository (no *zip archive allowed) (1 points).	

Fig 1. Project requirements

[1] the instruction and requirements provided on the enauczanie platform.

2. Summary of the tasks realized

My work on the project resulted in the creation of two applications - the auxiliary application (used for RSA key generation, from now on called “AuxiliaryApp” and the main one (handling PDF documents signing and their later validation, from now on called “Solution”).

As required in this project’s instruction, the AuxiliaryApp lets User A generate a pair of RSA keys 4096 bits in length, and saves the private key to the attached pendrive. The Solution then automatically detects this pendrive, loads the private key and starts the signing process, resulting in a PAdES-signed PDF file. Later on, User B can use User A’s public key to validate this signature.

Both of the applications’ documentations were generated using Doxygen and added at the end of this report.

3. AuxiliaryApp

3.1. Short description of the AuxiliaryApp

I prepared an auxiliary app (according to the requirements given in the project instruction). Its implemented functionalities are as follows:

- generation of a pair of 4096b RSA keys,
- obtaining a numerical PIN from the user,
- hashing of the private key with the AES algorithm, using 256b long SHA256 hash of said PIN,
- writing the encrypted private key to an attached pendrive,
- writing the public key to the project root directory,
- generation of a X.509 certificate (necessary for public key validation during signing),
- detection of a pendrive's presence/lack thereof,
- a GUI showing the user stages of execution taking place.

3.2. Simplified block diagram of the AuxiliaryApp

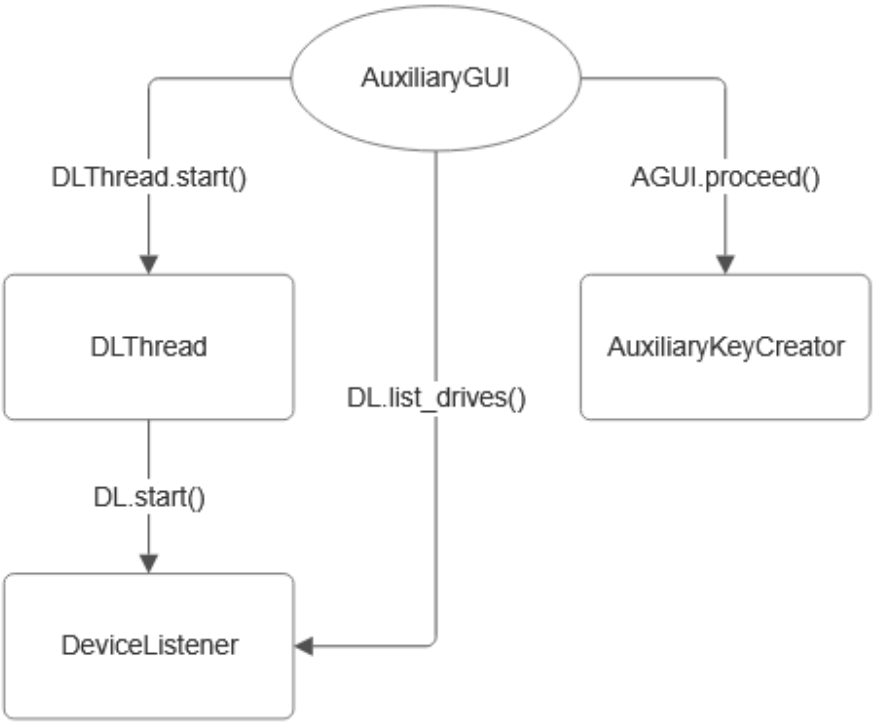


Fig 2. Simplified block diagram of the AuxiliaryApp

Explanation:

The singular point of entry to the AuxiliaryApp is the AuxiliaryGUI, which handles all the operations related to communicating with the user. When the user chooses to start the generation and encryption process, this class starts its proceed() method, which in turn invokes all the necessary methods of the AuxiliaryKeyCreator class, while also relaying the operation's progress to the user.

To determine whether the destination of encrypted private key - the pendrive - is currently available, the AuxiliaryGUI starts an additional DLThread (a descendant of the Thread class specially suited for device detection) with an instance of DeviceListener class inside. In case of an external device being attached/detached from the computer, it checks whether it is the desired pendrive, and changes the GUI status if so.

The AuxiliaryGUI instance also indirectly calls the list_devices() method of DeviceListener during setup phase, to determine the initial state of the pendrive.

3.3. Description of the most important methods

3.3.1. AuxiliaryGUI::on_devices_changed()

```
1 usage   ReadySetGet
def on_devices_changed(self):
    self._d_drive_comm.setText("Sprawdzam zmiany urzadzen zewnetrznych...")
    self._d_drive_comm.repaint()
    self._button.setEnabled(False)
    self._button.repaint()
    is_found = self.find_d_drive()
    if not self._is_d_drive_connected and is_found:
        self._is_d_drive_connected = True
        self._d_drive_comm.setText("Pendrive jest podpiety")
        self._button.setEnabled(True)
    elif self._is_d_drive_connected and not is_found:
        self._is_d_drive_connected = False
        self._d_drive_comm.setText("Pendrive nie jest podpiety")
        self._button.setEnabled(False)

    self._d_drive_comm.repaint()
    self._button.repaint()
    self.repaint()
```

Fig 3. AuxiliaryGUI::on_devices_changed()

This is the callback method of DeviceListener, it checks for an appearance/disappearance of the desired pendrive ("find_d_drive()" method), and changes the enabling of the starting button ("self.*_button") and the shown message ("self.*_d_drive_comm") accordingly.

3.3.2. AuxiliaryGUI::proceed()

```
self.show_current_arrow(self._current_stage_nr)
key_dict = {
    'key_priv': None,
    'key_pub': None
}
a = Thread(target=self._key_creator.generate_rsa_keys, args=(key_dict, ))
a.start()
while a.is_alive():
    QtWidgets.QApplication.instance().processEvents()
a.join()
key_priv = key_dict['key_priv']
key_pub = key_dict['key_pub']
self.set_texts()
time.sleep(0.5)

self.show_current_arrow(self._current_stage_nr)
pin_hash = self._key_creator.hash_pin_with_sha256(pin)
self.set_texts()
time.sleep(0.5)
```

Fig 4. Fragment of AuxiliaryGUI::proceed()

This method is the event function of the starting button, and it invokes all the required generation/hashing/encryption methods in the right order. To keep control during the lengthy generation stage, this method starts an additional Thread and continues to process requests from the user.

3.3.3. AuxiliaryKeyCreator::generate_rsa_keys()

```
1 usage   ReadySetGet
def generate_rsa_keys(self, arg):
    keypair = RSA.generate(self._constants.LENGTH_OF_RSA_KEY)
    arg['key_priv'] = keypair.export_key(format=self._constants.KEY_FORMAT)
    arg['key_pub'] = keypair.public_key().export_key(format=self._constants.KEY_FORMAT)
```

Fig 5. AuxiliaryKeyCreator::generate_rsa_keys()

This method uses the pycryptodome library to generate the RSA keypair of the given length (4096 bits in this case). Afterwards, it exports both of the keys to the DER format.

3.3.4. DeviceListener::_on_message(int, int, int, int)

```
1 usage   ReadySetGet
def _on_message(self, hwnd: int, msg: int, wparam: int, lparam: int):
    if msg != win32con.WM_DEVICECHANGE:
        return 0
    event, description = self.WM_DEVICECHANGE_EVENTS[wparam]
    if event in ('DBT_DEVICEREMOVECOMPLETE', 'DBT_DEVICEARRIVAL'):
        self.on_change()
    return 0
```

Fig 6. DeviceListener::_on_message(int, int, int, int)

This method uses the Windows operating system's window mechanisms to intercept messages connected to attaching/detaching a pendrive ("DBT_DEVICEREMOVECOMPLETE" and "DBT_DEVICEARRIVAL" of the "WM_DEVICECHANGE" type). It then invokes the provided callback method (in this case - AuxiliaryGUI::on_devices_changed()).

3.3.5. AuxiliaryKeyCreator::gen_cert()

```
1 usage   new *
def gen_cert(self):

    timestamp_epoch_time_start = 0
    key = crypto.load_publickey(crypto.FILETYPE_PEM,
                                open("C:/Studia/BSK/ProjektBSK/AuxiliaryApp/ProjectBSKPublicKey.pem").read())
    keyPriv = crypto.load_privatekey(crypto.FILETYPE_PEM,
                                     open("D:/ProjectBSKPrivateKey.pem").read(), self._pin_hash.digest())
    timestamp_epoch_time_end = 10 * 365 * 24 * 60 * 60
    cert_sign = crypto.X509()
    cert_sign.get_subject().CN = "Adam Zarzycki 193243"
    cert_sign.set_serial_number(int(time.time()))
    cert_sign.gmtime_adj_notBefore(timestamp_epoch_time_start)
    cert_sign.gmtime_adj_notAfter(timestamp_epoch_time_end)
    cert_sign.set_issuer(cert_sign.get_subject())
    cert_sign.set_pubkey(key)
    cert_sign.add_extensions([crypto.X509Extension(b"keyUsage", False, b"digitalSignature,nonRepudiation")])
    cert_sign.sign(keyPriv, "sha256")
    sign_cert = crypto.dump_certificate(crypto.FILETYPE_PEM, cert_sign)

    with open("certyfikat.pem", "wb") as certfile:
        certfile.write(sign_cert)
```

Fig 7. AuxiliaryKeyCreator::gen_cert()

This method uses the pyOpenSSL cryptography library to generate a new self-signed X.509 certificate for the future signing process. It is called after writing the keys to their respective files, so it loads them back (while decrypting the private key with the passphrase provided in the third parameter). It sets the subject's name ("get_subject().CN"), validity period (to 10 years, variable "timestamp_epoch_time_end"), issuer and certified public key. It also sets the necessary KeyUsage extension to digital signatures and non-repudiation capabilities. Lastly, it saves the certificate in the "certyfikat.pem" file.

3.4. Tests summary

- key generation without the pendrive attached - impossible, the "Start generation" button is not enabled,
- non-numerical value given as PIN - the request is repeated until a numerical value is provided,
- no key/certificate files present - new files are created in the paths intended,
- old key/certificate files present - files are overwritten with newly-generated values.

3.5. Used technology

- operating system - Windows 10,
- programming language - Python 3.11,
- GUI - PySide6,
- generation/hashing/encryption - pycryptodome 3.21.0,
- certificate generation - pyopenssl 25.1.0,
- device recognition - win32api,
- threading - threading.Thread.

4. Solution

4.1. Short description of the Solution

I prepared the main app of the project, according to the requirements provided in the project instruction. Its functionalities are as follows:

- automatic pendrive/private key detection,
- automatic loading of the private key to the application,
- obtaining a numerical PIN from the user,
- checking correctness of the PIN provided,
- decrypting the private key with a SHA256 hash of the PIN provided,
- letting User A choose the PDF file to be signed,
- signing the file with a PAdES qualified signature,
- saving the signed file,
- letting User B choose the PDF file which signature is to be verified,
- verifying the given file with User A's public key,
- a GUI showing the Users stages of execution taking place, and the result of the verification process.

4.2. Simplified block diagram of the Solution

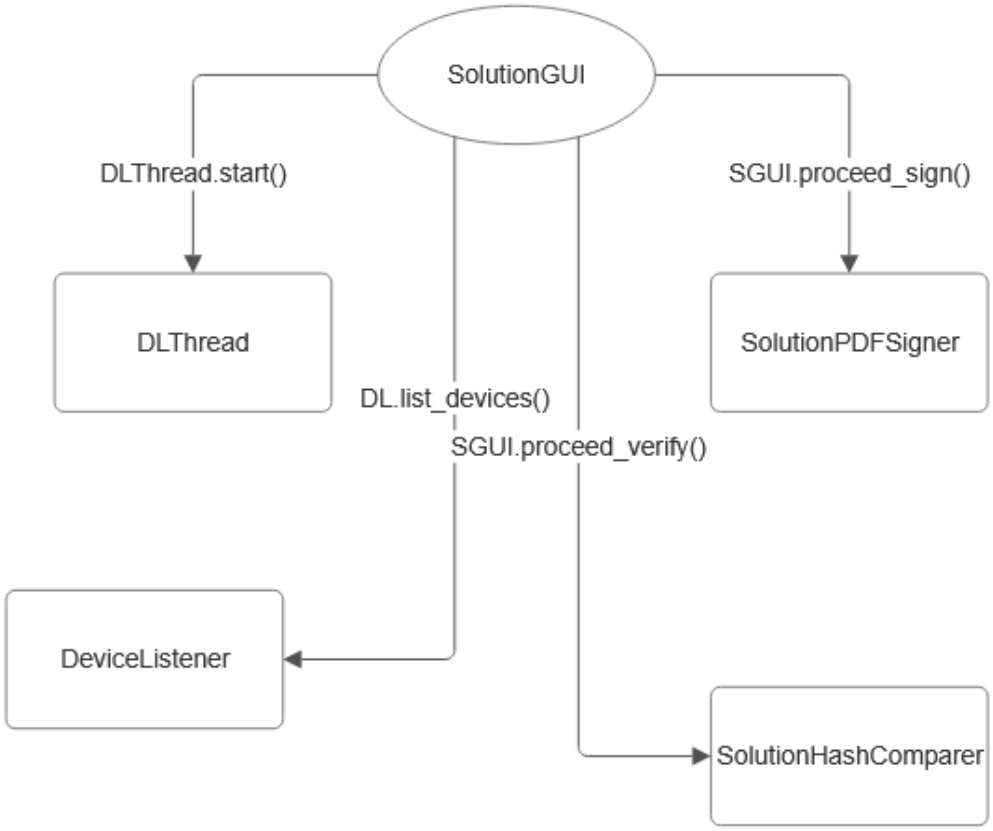


Fig 8. Simplified block diagram of the Solution

Explanation:

The Solution has a structure rather similar to the AuxiliaryApp. The SolutionGUI, DLThread, and DeviceListener have mostly the same behaviour as their auxiliary counterparts, with a few exceptions:

- SolutionGUI checks not only for a pendrive, but also for the presence of a private key on it,
- SolutionGUI automatically starts the signing process if a pendrive with a private key is detected, also on the start of the application.

The SolutionPDFSigner is invoked automatically by the SolutionGUI class to sign the chosen .pdf document (or on User's demand - as the signing process can also be invoked manually by them). The resulting PDF file with a signature is then saved in the place of its predecessor.

The SolutionHashComparer is invoked by the SolutionGUI manually, after receiving a command from the User. It lets them choose the PDF to be verified and returns to the SolutionGUI the result of this verification, which shows it to the User afterwards.

4.3. Description of the most important methods

4.3.1. SolutionGUI::proceed_verify()

```
self.show_current_arrow(self._current_stage_nr)
valid = self._hash_comparer.verify()
if valid == -1:
    self.generation_stages_init()
    self._button_sign.setEnabled(True if self._is_d_drive_connected else False)
    self._button_sign.repaint()

    self._button_verify.setEnabled(True)
    self._button_verify.repaint()
    self._result_comm.setText("Plik nie jest podpisany")
    return
self.set_texts_verify()
time.sleep(0.5)
```

Fig 9. SolutionGUI:proceed_verify() (fragment)

This method controls the flow of the verification process, by systematically calling the right methods of the “SolutionHashComparer” class. It works very similarly for every stage of this process, so on the listing only a logical fragment is shown.

First, it sets the current stage indicator by calling “show_current_arrow()” method. Then, it calls the “verify()” method of “SolutionHashComparer” class, which compares the signature with its supposed value. Then, it processes the return value, and changes the visible state of the application accordingly.

4.3.2. SolutionGUI::proceed_sign()

```
self.show_current_arrow(self._current_stage_nr)
was_good_pin = self._signer.decrypt()
if not was_good_pin:
    self.generation_stages_init()
    self._button_sign.setEnabled(True if self._is_d_drive_connected else False)
    self._button_sign.repaint()

    self._button_verify.setEnabled(True)
    self._button_verify.repaint()
    self._result_comm.setText("Niepoprawny klucz")
    return
self.set_texts_sign()
time.sleep(0.5)
```

Fig 10. SolutionGUI::proceed_sign() (fragment)

This method has the exact same structure as the "proceed_verify()" method described earlier, but it focuses on controlling the signing process flow instead. The fragment provided shows a call to "decrypt" method, which, besides decrypting the private key with the PIN hash, returns the result of a decryption check (whether it was the same PIN that was used for encrypting the key), so the GUI can adapt, and eventually stop the process.

4.3.3. SolutionPDFSigner::sign()

```
1 usage
def sign(self):

    with open(self._file_to_sign_path + self._file_to_sign, 'rb') as inf:
        w = IncrementalPdfFileWriter(inf, strict=False)
        with open('../pdfs/signed'+self._file_to_sign, 'wb') as outf:
            signers.sign_pdf(
                w, signature_meta=self._signature_meta, signer=self._cms_signer,
                output=outf
            )

    os.remove(self._file_to_sign_path + self._file_to_sign)
```

Fig 11. SolutionPDFSigner::sign()

This method uses the pyHanko digital signature library to sign the provided PDF file ("inf") to an output file ("outf"), using configuration from an earlier call to the "SolutionPDFSigner::prepare_file()" method. It also removes the original file.

4.3.4. SolutionHashComparer::verify()

```
1 usage
def verify(self):

    with open(self._file_path + self._file_name, 'rb') as doc:
        self._r = PdfFileReader(doc, strict=False)
        if len(self._r.embedded_signatures) == 0:
            return -1
        self._sig = self._r.embedded_signatures[0]
        status = validate_pdf_signature(self._sig, self._vc)
        print(status.pretty_print_details())
        if status.bottom_line:
            return 1

        return 0
```

Fig 12. SolutionHashComparer::verify()

This method opens the chosen signed PDF file and extracts its signature, after which it uses the pyHanko method “validate_pdf_signature()” to hash the provided file and compare it with the signature. Then, it checks the overall validity of the signature (“status.bottom_line”), and returns accordingly.

It is worth mentioning that if there is no signature in the chosen file, this method returns with a different value, so the “SolutionGUI” class can show an adequate message.

4.4. Testing

- pendrive attached on program start - automatically detected, key loaded,
- pendrive not attached on program start - signing impossible,
- pendrive attached later - automatically detected, key loaded,
- key not present on pendrive - signing impossible,
- non-numerical PIN - the request is repeated until a numerical value is provided,
- wrong PIN (not the one the key was encrypted with) - the process is stopped with an “Incorrect key” message,

- choosing of a non-PDF file - impossible, only PDF files are shown,
- no file chosen - the process is stopped with a “No file chosen” message,
- signing an already-signed PDF - the process is stopped with a “File is already signed” message,
- verifying a file without a signature - the process is stopped with a “File is not signed” message,
- **verifying a file with a correct signature -** the process ends with a “Verification succeeded - the document is correct”,
- **verifying a file that was changed after signing -** the process ends with a “Verification failed - document had been changed” message.

4.5. Used technology

- operating system - Windows 10,
- programming language - Python 3.11,
- GUI - PySide6,
- hashing/decryption - pycryptodome 3.23.0,
- signing/verifying/PDF writing/reading - pyhanko 0.29.0 / pyhanko-certvalidator 0.27.0,
- device recognition - win32api,
- threading - threading.Thread.

5. Link to Github repository

Link: https://github.com/ReadySetGet/Projekt-BSK

6. Literature

- the basic skeleton of DeviceListener class: https://abdus.dev/posts/python-monitor-usb/,
- pycryptodome online documentation: https://www.pycryptodome.org/,
- PySide6 online documentation: https://doc.qt.io/qtforpython-6/index.html,
- pyOpenSSL online documentation: https://www.pyopenssl.org/en/stable/index.html,
- pyHanko online documentation: https://pyhanko.readthedocs.io/en/latest/lib-guide/index.html,
- the instruction and requirements provided on the enauczanie platform.

7. Generated documentation - Doxygen

The generated documentation is attached below.

ProjectBSK

1.0

Generated by Doxygen 1.14.0

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Namespace Documentation

4.1 AuxiliaryGUI Namespace Reference

It revolves around the user interface of the auxiliary application, controls the process of key generation, and enables it based on feedback from DeviceListener.

Classes

- class AuxiliaryGUI

  The auxiliary app GUI class.

4.1.1 Detailed Description

It revolves around the user interface of the auxiliary application, controls the process of key generation, and enables it based on feedback from DeviceListener.

4.2 AuxiliaryKeyCreator Namespace Reference

It provides all the functionalities necessary from the technical perspective to execute the key generation proccess.

Classes

- class AuxiliaryKeyCreator

  The generator class.

4.2.1 Detailed Description

It provides all the functionalities necessary from the technical perspective to execute the key generation proccess.

4.3 DeviceListener Namespace Reference

The listener package for detecting changes in drives' configuration, based on `this article`, and configured to fit the project requirements.

Classes

- class DeviceListener

  Main listener class.
- class Drive

  A dataclass for storing found drives' information.

4.3.1 Detailed Description

The listener package for detecting changes in drives' configuration, based on `this article`, and configured to fit the project requirements.

4.4 DLThread Namespace Reference

A wrapper for python's `threading.Thread` class, with added DeviceListener stopping capabilities.

Classes

- class DLThread

  The main class, inheriting from `threading.Thread` python class.

4.4.1 Detailed Description

A wrapper for python's `threading.Thread` class, with added DeviceListener stopping capabilities.

4.5 main Namespace Reference

The entrypoint to the auxiliary app.

Variables

- **app** = QtWidgets.QApplication([ ])
- **widget** = AuxiliaryGUI()
- **sign_on_start** = threading.Thread(widget.sign_if_pendrive_on_start())

4.5.1 Detailed Description

The entrypoint to the auxiliary app.

The entrypoint to the main app.

It starts the app's widget.

It starts the widget and a thread checking for a pendrive with a key.

4.6 SolutionGUI Namespace Reference

It revolves around the user interface of the main application, controls the processes of signing/verification, and enables them based on feedback from DeviceListener.

Classes

- class SolutionGUI
  The main app GUI class.

4.6.1 Detailed Description

It revolves around the user interface of the main application, controls the processes of signing/verification, and enables them based on feedback from DeviceListener.

4.7 SolutionHashComparer Namespace Reference

It realizes all the functionalities needed for the verification process.

Classes

- class SolutionHashComparer
  The verifier class.

4.7.1 Detailed Description

It realizes all the functionalities needed for the verification process.

4.8 SolutionPDFSigner Namespace Reference

It provides all the functionalities necessary from the technical perspective to execute the signing proccess.

Classes

- class SolutionPDFSigner
  The signer class.

4.8.1 Detailed Description

It provides all the functionalities necessary from the technical perspective to execute the signing proccess.

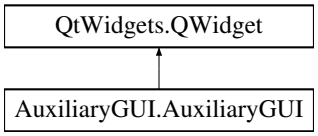
Chapter 5

Class Documentation

5.1 AuxiliaryGUI.AuxiliaryGUI Class Reference

The auxiliary app GUI class.

Inheritance diagram for AuxiliaryGUI.AuxiliaryGUI:



Public Member Functions

- __init__ (self)

  Constructor.
- generation_stages_init (self)

  It brings back the default settings of the app.
- proceed (self)

  Controller of the generation process.
- ask_for_pin (self)

  Getting pin from the user.
- **show_current_arrow** (self, idx)

  Changes position of the arrow showing currently executed stage to the next stage.
- **set_texts** (self)

  A `proceed()` helper function, changing the messages of the executed stages to their "done" counterparts, and marking them as such.
- find_d_drive (self)

  Checking the pendrive's status.
- on_devices_changed (self)

  Checking the device setup changes.
- **end_listening** (self)

  A method for ending the listener's thread.

Public Attributes

- **proceed**
- **end_listening**

Protected Attributes

- **_constants** = Constants(NR_OF_STAGES=6)
- int **_current_stage_nr** = 0
- **_key_creator** = AuxiliaryKeyCreator()
- **_listener** = DeviceListener(on_change=self.on_devices_changed)
- bool **_is_d_drive_connected** = self.find_d_drive()
- **_d_drive_comm** = QtWidgets.QLabel("Pendrive nie jest podpiety" if not self._is_d_drive_connected else "Pendrive jest podpiety")
- **_ending_comm** = QtWidgets.QLabel("")
- **_button** = QtWidgets.QPushButton("Rozpocznij generowanie")
- **_button_close** = QtWidgets.QPushButton("Wyjdź z programu")
- **_layout** = QtWidgets.QVBoxLayout(self)
- **_info** = QtWidgets.QWidget(self)
- **_grid** = QtWidgets.QGridLayout(self._info)
- list **_start_texts** = ["Podaj PIN:", "Rozpoczęto generacje kluczy", "Rozpoczęto hashowanie PIN-u", "Rozpoczęto szyfrowanie klucza AES-em", "Rozpoczęto zapisywanie klucza publicznego na dysku", "Rozpoczęto zapisywanie klucza prywatnego na pendrivie"]
- list **_end_texts** = ["Pobrano PIN", "Zakończono generacje kluczy", "Zakończono hashowanie PIN-u", "Zakończono szyfrowanie klucza AES-em", "Zakończono zapisywanie klucza publicznego na dysku", "Zakończono zapisywanie klucza prywatnego na pendrivie"]
- str **_ending_comm_text** = "Wykonano wszystkie zadania"
- list **_stage_comms** = [ ]
- list **_stage_checks** = [ ]
- list **_arrows** = [ ]
- **_listenerThread** = DLThread(target=self._listener.start)

5.1.1 Detailed Description

The auxiliary app GUI class.

It realizes all the functionalities of this package.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 __init__()

```
AuxiliaryGUI.AuxiliaryGUI.__init__ (
                self)
```

Constructor.

It Initializes all the widget's elements, used constants, and places them in the layout.

5.1.3 Member Function Documentation

5.1.3.1 ask_for_pin()

```
AuxiliaryGUI.AuxiliaryGUI.ask_for_pin (
              self)
```

Getting pin from the user.

It shows a `QInputDialog` instance to the user, and checks whether the pin provided has numerical value.

5.1.3.2 find_d_drive()

```
AuxiliaryGUI.AuxiliaryGUI.find_d_drive (
              self)
```

Checking the pendrive's status.

It calls the listener's `list_drives()` function, and checks if the pendrive (with a private key present) is amongst them.

Returns

(bool) whether the pendrive with a key was found or

5.1.3.3 generation_stages_init()

```
AuxiliaryGUI.AuxiliaryGUI.generation_stages_init (
              self)
```

It brings back the *default settings* of the app.

It brings back the *default settings* of the app, clearing all the labels and enabling the buttons accordingly (based on whether the pendrive is available).

5.1.3.4 on_devices_changed()

```
AuxiliaryGUI.AuxiliaryGUI.on_devices_changed (
              self)
```

Checking the device setup changes.

It's called by DeviceListener class when a change in drives' configuration has been detected. It calls the `find_d_drive()` method to ascertain the desired pendrive's presence, and if so, it loads the encrypted private key (via initializing the AuxiliaryKeyCreator class), and starts the signing process.

5.1.3.5 proceed()

```
AuxiliaryGUI.AuxiliaryGUI.proceed (
              self)
```

Controller of the generation process.

Controller of the generation process. It resets the app with `generation_stages_init()`, and executes all the steps necessary for the generation process to succeed, by invoking the adequate methods of AuxiliaryKeyCreator class. It shows the user all relevant progress messages, and changes the position of the progress arrow. Lastly, it presents the result of actions taken to the user.

The documentation for this class was generated from the following file:

- AuxiliaryApp/AuxiliaryGUI.py

5.2 AuxiliaryKeyCreator.AuxiliaryKeyCreator Class Reference

The generator class.

Public Member Functions

- __init__ (self)
  Constructor.
- generate_rsa_keys (self, arg)
  Main generator method.
- hash_pin_with_sha256 (self, pin)
  It hashes the pin provided with SHA256.
- cipher_key_with_aes (self, pin_hash, key_priv)
  It encrypts the private key with AES256 algorithm.
- **write_public_key_to_file** (self, pubkey)
  It writes the public key to a .pem file.
- write_private_key_to_pendrive (self, key_priv_with_aes)
  It writes the private key to a .pem file.
- **gen_cert** (self)
  It generates a certificate based on the generated public key, needed for later PAdES digital signature, and saves it to a file.

Protected Attributes

- _constants
- **_keypair** = RSA.generate(self._constants.LENGTH_OF_RSA_KEY)
- **_pin_hash** = SHA256.new(bytes(pin))

5.2.1 Detailed Description

The generator class.

It realizes all the functionalities of this package.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 __init__()

```
AuxiliaryKeyCreator.AuxiliaryKeyCreator.__init__ (
            self)
```

Constructor.

It sets the constants used.

5.2.3 Member Function Documentation

5.2.3.1 cipher_key_with_aes()

```
AuxiliaryKeyCreator.AuxiliaryKeyCreator.cipher_key_with_aes (
            self,
            pin_hash,
            key_priv)
```

It encrypts the private key with AES256 algorithm.

Parameters

	pin_hash	(bytes): hash of the pin used as AES passphrase

Returns

(bytes) the encrypted private key

5.2.3.2 generate_rsa_keys()

```
AuxiliaryKeyCreator.AuxiliaryKeyCreator.generate_rsa_keys (
            self,
            arg)
```

Main generator method.

It generates the RSA keys and exports them to .pem format

Parameters

	arg	({**setitem**}): a way of returning the keys to AuxiliaryGUI.

5.2.3.3 hash_pin_with_sha256()

```
AuxiliaryKeyCreator.AuxiliaryKeyCreator.hash_pin_with_sha256 (
            self,
            pin)
```

It hashes the pin provided with SHA256.

Parameters

	pin	(int): the pin provided

Returns

(bytes) hash of the pin

5.2.3.4 write_private_key_to_pendrive()

```
AuxiliaryKeyCreator.AuxiliaryKeyCreator.write_private_key_to_pendrive (
              self,
              key_priv_with_aes)
```

It writes the private key to a .pem file.

Parameters

	key_priv_with_aes	(bytes): the encrypted private key

5.2.4 Member Data Documentation

5.2.4.1 _constants

```
AuxiliaryKeyCreator.AuxiliaryKeyCreator._constants  [protected]
```

Initial value:

```
=  Constants(LENGTH_OF_RSA_KEY=4096, KEY_FORMAT='PEM', CIPHER_MODE=AES.MODE_CBC,
      PATH_FOR_TO__PUBLIC_KEY_FILE=
                                        "C:/Studia/BSK/ProjektBSK/AuxiliaryApp")
```

The documentation for this class was generated from the following file:

- AuxiliaryApp/AuxiliaryKeyCreator.py

5.3 DeviceListener.DeviceListener Class Reference

Main listener class.

Public Member Functions

- __init__ (self, Callable[[ ], None] on_change)
  Constructor.
- start (self)
  Entry point of the class.
- **close** (self)
  Closes the window.
- __init__ (self, Callable[[ ], None] on_change)
  Constructor.
- start (self)
  Entry point of the class.
- **close** (self)
  Closes the window.

Static Public Member Functions

- List[Drive] list_drives ()

  Lists all attached drives, with the detail level provided by Drive class.
- List[Drive] list_drives ()

  Lists all attached drives, with the detail level provided by Drive class.

Public Attributes

- **on_change** = on_change
- **hwnd** = self._create_window()

Static Public Attributes

- dict WM_DEVICECHANGE_EVENTS

Protected Member Functions

- _create_window (self)

  Creates a new win32 message window.
- _on_message (self, int hwnd, int msg, int wparam, int lparam)

  The method called after a new message arrives.
- _create_window (self)

  Creates a new win32 message window.
- _on_message (self, int hwnd, int msg, int wparam, int lparam)

  The method called after a new message arrives.

5.3.1 Detailed Description

Main listener class.

It realizes all of ths package's functionalities.

Attributes:

- WM_DEVICECHANGE_EVENTS: a dictionary of event codes with their description

5.3.2 Constructor & Destructor Documentation

5.3.2.1 __init__() [1/2]

```
DeviceListener.DeviceListener.__init__ (
               self,
              Callable[[], None] on_change)
```

Constructor.

Sets the method called.

Parameters

	on_change	(Callable[[], None]): method to be called

5.3.2.2 __init__() [2/2]

```
DeviceListener.DeviceListener.__init__ (
              self,
             Callable[[], None] on_change)
```

Constructor.

Sets the method called.

Parameters

	on_change	(Callable[[], None]): method to be called

5.3.3 Member Function Documentation

5.3.3.1 _create_window() [1/2]

```
DeviceListener.DeviceListener._create_window (
              self)  [protected]
```

Creates a new win32 message window.

Returns

(int) handler for the new window

5.3.3.2 _create_window() [2/2]

```
DeviceListener.DeviceListener._create_window (
              self)  [protected]
```

Creates a new win32 message window.

Returns

(int) handler for the new window

5.3.3.3 _on_message() [1/2]

```
DeviceListener.DeviceListener._on_message (
              self,
             int hwnd,
             int msg,
             int wparam,
             int lparam)  [protected]
```

The method called after a new message arrives.

It checks whether an important change occurred, and calls the provided `on_change()` method.

Parameters

hwnd	(int): handler for the window
msg	(int): the processed message
wparam	(int): the higher part of the message word
lparam	(int): the lower part of the message word

Returns

0 - method finished correctly

5.3.3.4 _on_message() [2/2]

```
DeviceListener.DeviceListener._on_message (
              self,
             int hwnd,
             int msg,
             int wparam,
             int lparam)  [protected]
```

The method called after a new message arrives.

It checks whether an important change occurred, and calls the provided `on_change()` method.

Parameters

hwnd	(int): handler for the window
msg	(int): the processed message
wparam	(int): the higher part of the message word
lparam	(int): the lower part of the message word

Returns

0 - method finished correctly

5.3.3.5 list_drives() [1/2]

```
List[Drive] DeviceListener.DeviceListener.list_drives ()  [static]
```

Lists all attached drives, with the detail level provided by Drive class.

Returns

(ListDrive]) a list of drives

5.3.3.6 list_drives() [2/2]

```
List[Drive] DeviceListener.DeviceListener.list_drives ()  [static]
```

Lists all attached drives, with the detail level provided by Drive class.

Returns

(ListDrive]) a list of drives

5.3.3.7 start() [1/2]

```
DeviceListener.DeviceListener.start (
               self)
```

Entry point of the class.

Calls for a new window and starts taking in messages.

5.3.3.8 start() [2/2]

```
DeviceListener.DeviceListener.start (
               self)
```

Entry point of the class.

Calls for a new window and starts taking in messages.

5.3.4 Member Data Documentation

5.3.4.1 WM_DEVICECHANGE_EVENTS

```
dict DeviceListener.DeviceListener.WM_DEVICECHANGE_EVENTS  [static]
```

Initial value:

```
=  {
        0x0019: ('DBT_CONFIGCHANGECANCELED', 'A request to change the current configuration (dock or undock)
      has been canceled.'),
        0x0018: ('DBT_CONFIGCHANGED', 'The current configuration has changed, due to a dock or undock.'),
        0x8006: ('DBT_CUSTOMEVENT', 'A custom event has occurred.'),
        0x8000: ('DBT_DEVICEARRIVAL', 'A device or piece of media has been inserted and is now available.'),
        0x8001: ('DBT_DEVICEQUERYREMOVE', 'Permission is requested to remove a device or piece of media. Any
      application can deny this request and cancel the removal.'),
        0x8002: ('DBT_DEVICEQUERYREMOVEFAILED', 'A request to remove a device or piece of media has been
      canceled.'),
        0x8004: ('DBT_DEVICEREMOVECOMPLETE', 'A device or piece of media has been removed.'),
        0x8003: ('DBT_DEVICEREMOVEPENDING', 'A device or piece of media is about to be removed. Cannot be
      denied.'),
        0x8005: ('DBT_DEVICETYPESPECIFIC', 'A device-specific event has occurred.'),
        0x0007: ('DBT_DEVNODES_CHANGED', 'A device has been added to or removed from the system.'),
        0x0017: ('DBT_QUERYCHANGECONFIG', 'Permission is requested to change the current configuration (dock
      or undock).'),
        0xFFFF: ('DBT_USERDEFINED', 'The meaning of this message is user-defined.'),
    }
```

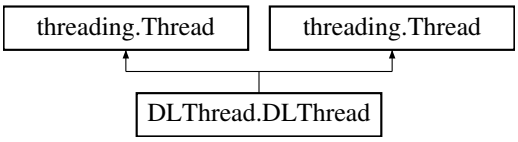
The documentation for this class was generated from the following files:

- AuxiliaryApp/DeviceListener.py
- Solution/DeviceListener.py

5.4 DLThread.DLThread Class Reference

The main class, inheriting from `threading.Thread` python class.

Inheritance diagram for DLThread.DLThread:



Public Member Functions

- **__init__** (self, *args, **keywords)

  Constructor.
- **kill** (self)

  It stops the associated listener by sending an appropriate win32api message.
- **__init__** (self, *args, **keywords)

  Constructor.
- **kill** (self)

  It stops the associated listener by sending an appropriate win32api message.

Public Attributes

- **ident**

5.4.1 Detailed Description

The main class, inheriting from `threading.Thread` python class.

The documentation for this class was generated from the following files:

- AuxiliaryApp/DLThread.py
- Solution/DLThread.py

5.5 DeviceListener.Drive Class Reference

A dataclass for storing found drives' information.

Public Member Functions

- bool **is_removable** (self)

  Whether the drive is removable or not.
- bool **is_removable** (self)

  Whether the drive is removable or not.

Public Attributes

- str **drive_type** = 'Removable Disk'

5.5.1 Detailed Description

A dataclass for storing found drives' information.

Attributes:

- letter: drive's letter
- label: drive's label
- drive_type: drive's type

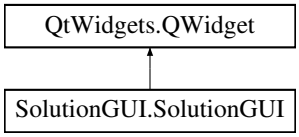
The documentation for this class was generated from the following files:

- AuxiliaryApp/DeviceListener.py
- Solution/DeviceListener.py

5.6 SolutionGUI.SolutionGUI Class Reference

The main app GUI class.

Inheritance diagram for SolutionGUI.SolutionGUI:



Public Member Functions

- __init__ (self)
  Constructor.
- generation_stages_init (self)
  It brings back the default settings of the app.
- proceed_sign (self)
  Controller of the signing process.
- proceed_verify (self)
  Controller of the verifying process.
- ask_for_pin (self)
  Getting pin from the user.
- **show_current_arrow** (self, idx)
  Changes position of the arrow showing currently executed stage to the next stage.
- **set_texts_sign** (self)

A `proceed_sign()` helper function, changing the messages of the executed stages to their "done" counterparts, and marking them as such.

- **set_texts_verify** (self)

  A `proceed_verify()` helper function, changing the messages of the executed stages to their "done" counterparts, and marking them as such.

- find_d_drive (self)

  Checking the pendrive's status.

- on_devices_changed (self)

  Checking the device setup changes.

- **end_listening** (self)

  A method for ending the listener's thread.

- **sign_if_pendrive_on_start** (self)

  A method for a thread to check whether the app should load the key and start the signing proccess on execution.

Public Attributes

- **proceed_sign**
- **proceed_verify**
- **end_listening**

Protected Attributes

- **_constants** = Constants(NR_OF_STAGES_SIGNING=6, NR_OF_STAGES_VERIFYING=3)
- int **_current_stage_nr** = 0
- SolutionPDFSigner **_signer** = None
- **_hash_comparer** = SolutionHashComparer()
- **_listener** = DeviceListener(on_change=self.on_devices_changed)
- bool **_is_d_drive_connected** = self.find_d_drive()
- **_d_drive_comm** = QtWidgets.QLabel("Pendrive nie jest podpięty" if not self._is_d_drive_connected else "Pendrive jest podpięty, klucz został pobrany")
- **_ending_comm** = QtWidgets.QLabel("")
- **_result_comm** = QtWidgets.QLabel("")
- **_button_sign** = QtWidgets.QPushButton("Rozpocznij podpisywanie")
- **_button_verify** = QtWidgets.QPushButton("Rozpocznij weryfikacje")
- **_button_close** = QtWidgets.QPushButton("Wyjdź z programu")
- **_layout** = QtWidgets.QVBoxLayout(self)
- **_info** = QtWidgets.QWidget(self)
- **_grid** = QtWidgets.QGridLayout(self._info)
- list **_start_texts_sign** = ["Podaj PIN:", "Wybieranie dokumentu", "Rozpoczęto hashowanie PIN-u", "Rozpoczęto deszyfrowanie klucza", "Rozpoczęto przygotowanie dokumentu", "Rozpoczęto podpisywanie dokumentu"]
- list **_end_texts_sign** = ["Pobrano PIN", "Zakończono wybieranie dokumentu", "Zakończono hashowanie PIN-u", "Zakończono deszyfrowanie klucza", "Zakończono przygotowanie dokumentu", "Zakończono podpisywanie dokumentu"]
- list _start_texts_verify
- list _end_texts_verify
- str **_ending_comm_text** = "Wykonano wszystkie zadania"
- list **_stage_comms** = [ ]
- list **_stage_checks** = [ ]
- list **_arrows** = [ ]
- **_listenerThread** = DLThread(target=self._listener.start)

5.6.1 Detailed Description

The main app GUI class.

It realizes all the functionalities of this package.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 __init__()

```
SolutionGUI.SolutionGUI.__init__ (
              self)
```

Constructor.

It Initializes all the widget's elements, used constants, and places them in the layout.

5.6.3 Member Function Documentation

5.6.3.1 ask_for_pin()

```
SolutionGUI.SolutionGUI.ask_for_pin (
              self)
```

Getting pin from the user.

It shows a `QInputDialog` instance to the user, and checks whether the pin provided has numerical value.

5.6.3.2 find_d_drive()

```
SolutionGUI.SolutionGUI.find_d_drive (
              self)
```

Checking the pendrive's status.

It calls the listener's `list_drives()` function, and checks if the pendrive (with a private key present) is amongst them.

Returns

(bool) whether the pendrive with a key was found or

5.6.3.3 generation_stages_init()

```
SolutionGUI.SolutionGUI.generation_stages_init (
              self)
```

It brings back the *default settings* of the app.

It brings back the *default settings* of the app, clearing all the labels and enabling the buttons accordingly (based on whether the pendrive is available).

5.6.3.4 on_devices_changed()

```
SolutionGUI.SolutionGUI.on_devices_changed (
            self)
```

Checking the device setup changes.

It's called by DeviceListener class when a change in drives' configuration has been detected. It calls the `find_d_drive()` method to ascertain the desired pendrive's presence, and if so, it loads the encrypted private key (via initializing the SolutionPDFSigner class), and starts the signing process.

5.6.3.5 proceed_sign()

```
SolutionGUI.SolutionGUI.proceed_sign (
            self)
```

Controller of the signing process.

Controller of the signing process. It resets the app with `generation_stages_init()`, and executes all the steps necessary for the signing process to succeed, by invoking the adequate methods of SolutionPDFSigner class. It shows the user all relevant progress messages, and changes the position of the progress arrow. Lastly, it presents the result of actions taken to the user.

5.6.3.6 proceed_verify()

```
SolutionGUI.SolutionGUI.proceed_verify (
            self)
```

Controller of the verifying process.

Controller of the verifying process. It resets the app with `generation_stages_init()`, and executes all the steps necessary for the signing process to succeed, by invoking the adequate methods of SolutionHashComparer class. It shows the user all relevant progress messages, and changes the position of the progress arrow. Lastly, it presents the result of actions taken to the user.

5.6.4 Member Data Documentation

5.6.4.1 _end_texts_verify

```
list SolutionGUI.SolutionGUI._end_texts_verify  [protected]
```

Initial value:

```
=  ["Zakończono wybieranie dokumentu", "Zakończono pobieranie klucza publicznego",
                              "Zakończono weryfikację"]
```

5.6.4.2 _start_texts_verify

```
list SolutionGUI.SolutionGUI._start_texts_verify  [protected]
```

Initial value:

```
=  ["Wybieranie dokumentu", "Rozpoczęto pobieranie klucza publicznego",
                                "Rozpoczęto weryfikację"]
```

The documentation for this class was generated from the following file:

- Solution/SolutionGUI.py

5.7 SolutionHashComparer.SolutionHashComparer Class Reference

The verifier class.

Public Member Functions

- __init__ (self)
  Constructor.
- set_file (self, path, name)
  A setter for all pdf-related information.
- **set_public_key** (self)
  It loads the public key and its certificate from a file.
- verify (self)
  It validates the signature, based on the public key and certificate loaded by `set_public_key()` method It then prints the process' details to the console.

Protected Attributes

- _constants
- **_public_key** = None
- **_file_path** = None
- **_file_name** = None
- **_signature** = None
- **_hashed_file** = None
- **_vc** = ValidationContext(trust_roots=[root_cert])
- **_r** = PdfFileReader(doc, strict=False)
- **_sig** = self._r.embedded_signatures[0]

5.7.1 Detailed Description

The verifier class.

It realizes all the functionalities of this package.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 __init__()

```
SolutionHashComparer.SolutionHashComparer.__init__ (
            self)
```

Constructor.

It sets the constants used.

5.7.3 Member Function Documentation

5.7.3.1 set_file()

```
SolutionHashComparer.SolutionHashComparer.set_file (
            self,
            path,
            name)
```

A setter for all pdf-related information.

Parameters

	path	(str): path to the pdf, without its name
	file	(str): name of the pdf

5.7.3.2 verify()

```
SolutionHashComparer.SolutionHashComparer.verify (
            self)
```

It validates the signature, based on the public key and certificate loaded by `set_public_key()` method It then prints the process' details to the console.

Returns

1: the signature is valid

0: the signature is invalid

-1: the chosen file has no signature to verify

5.7.4 Member Data Documentation

5.7.4.1 _constants

```
SolutionHashComparer.SolutionHashComparer._constants  [protected]
```

Initial value:

```
=  Constants(LENGTH_OF_RSA_KEY=4096, KEY_FORMAT='PEM',
                                    PATH_FOR_PUBLIC_KEY="C:/Studia/BSK/ProjektBSK/AuxiliaryApp/ProjectBSKPublicKey.pem")
```

The documentation for this class was generated from the following file:

- Solution/SolutionHashComparer.py

5.8 SolutionPDFSigner.SolutionPDFSigner Class Reference

The signer class.

Public Member Functions

- __init__ (self)
  Constructor.
- set_file (self, path, file)
  A setter for all pdf-related information.
- hash_pin (self, pin)
  It hashes the pin provided.
- decrypt (self)
  It decrypts the private key the hash of the pin provided.
- prepare_file (self)
  It conducts all the necessary preparations before signing the chosen pdf: add a signature field to the pdf, sets the signature type to PAdES, and initializes an adequate PAdES signer object.
- **sign** (self)
  It signs and saves the pdf, using setting set in the `prepare_file()` method.

Protected Attributes

- _constants
- **_path_to_ske** = self._constants.PATH_TO_PRIVATE_KEY
- **_signing_key_encrypted** = file.read()
- **_signing_key** = None
- **_file_to_sign_path** = None
- **_file_to_sign** = None
- **_hashed_pin** = None
- **_hashed_file** = None
- **_signature** = None
- _cms_signer
- _signature_meta

5.8.1 Detailed Description

The signer class.

It realizes all the functionalities of this package.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 __init__()

```
SolutionPDFSigner.SolutionPDFSigner.__init__ (
            self)
```

Constructor.

It sets the used constants and loads the encrypted private key from a file.

5.8.3 Member Function Documentation

5.8.3.1 decrypt()

```
SolutionPDFSigner.SolutionPDFSigner.decrypt (
            self)
```

It decrypts the private key the hash of the pin provided.

Returns

(bool) whether the key was decrypted correctly or not (in other words, if the pin was correct)

5.8.3.2 hash_pin()

```
SolutionPDFSigner.SolutionPDFSigner.hash_pin (
            self,
            pin)
```

It hashes the pin provided.

Parameters

	pin	(int): pin

5.8.3.3 prepare_file()

```
SolutionPDFSigner.SolutionPDFSigner.prepare_file (
              self)
```

It conducts all the necessary preparations before signing the chosen pdf: add a signature field to the pdf, sets the signature type to PAdES, and initializes an adequate PAdES signer object.

Returns

1: the method succeeded

0: pdf has already been signed

-1: no writing permissions

5.8.3.4 set_file()

```
SolutionPDFSigner.SolutionPDFSigner.set_file (
              self,
              path,
              file)
```

A setter for all pdf-related information.

Parameters

	path	(str): path to the pdf, without its name :param file (str): name of the pdf

5.8.4 Member Data Documentation

5.8.4.1 _cms_signer

```
SolutionPDFSigner.SolutionPDFSigner._cms_signer  [protected]
```

Initial value:

```
=  signers.SimpleSigner.load(
            self._constants.PATH_TO_PRIVATE_KEY, "C:/Studia/BSK/ProjektBSK/AuxiliaryApp/certyfikat.pem",
            key_passphrase=self._hashed_pin.digest()
        )
```

5.8.4.2 _constants

```
SolutionPDFSigner.SolutionPDFSigner._constants  [protected]
```

Initial value:

```
=  Constants(LENGTH_OF_RSA_KEY=4096, KEY_FORMAT='PEM', CIPHER_MODE=AES.MODE_CBC,
                                  PATH_FOR_SIGNED_FILES="C:/Studia/BSK/ProjektBSK/Solution/",
      PATH_TO_PRIVATE_KEY="D:/ProjectBSKPrivateKey.pem")
```

5.8.4.3 _signature_meta

SolutionPDFSigner.SolutionPDFSigner._signature_meta [protected]

Initial value:

```
=  signers.PdfSignatureMetadata(
            field_name='Signature', md_algorithm='sha256',
            subfilter=SigSeedSubFilter.PADES,
            use_pades_lta=False
        )
```

The documentation for this class was generated from the following file:

- Solution/SolutionPDFSigner.py

Index