Security of Computer Systems

Project Report

Author:
Adam, Zarzycki, 193243

Version: 1.0

Versions

Version	Date	Description of changes
1.0	11.04.2025	Creation of the document

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. Project – control term

2.1. Short description of the AuxiliaryApp

To fulfill the requirements for the control term (as presented in point 1.2), I prepared an auxiliary application (called “AuxiliaryApp” from now on), which 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,
- detection of a pendrive’s presence/lack thereof,
- a GUI showing the user stages of execution taking place.

2.2. Simplified block diagram of the AuxiliaryApp

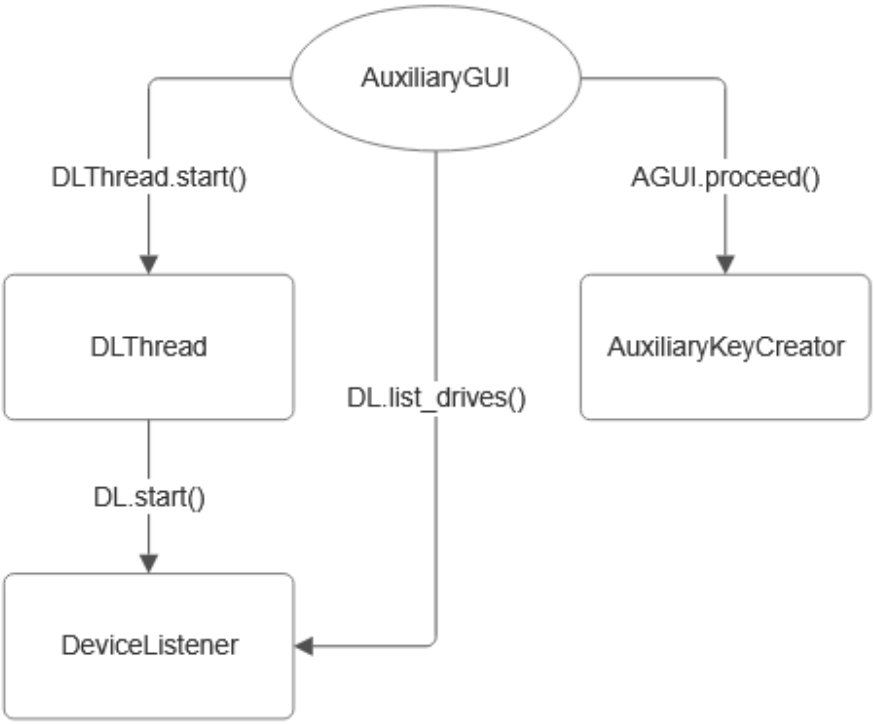

Fig 2. Simplified block diagram of the AuxiliaryApp

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.

2.3. Description of the most important methods

2.3.1. 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.

2.3.2. 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.

2.3.3. 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.

2.3.4. 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()).

2.4. Used technology

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

2.5. Link to Github repository

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

2.6. 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,
- the instruction and requirements provided on the enauczanie platform.

3. Project – Final term

3.1. Short description of the Solution

The main application (called "Solution" from now on) will detect the pendrive with the encrypted private key, use it to sign a .pdf file of the user's choice (after obtaining his or her PIN number). Also, another user will be able to verify the signature, by generating a hash of the signed document and comparing it with the original hash.

3.2. Simplified block diagram of the Solution

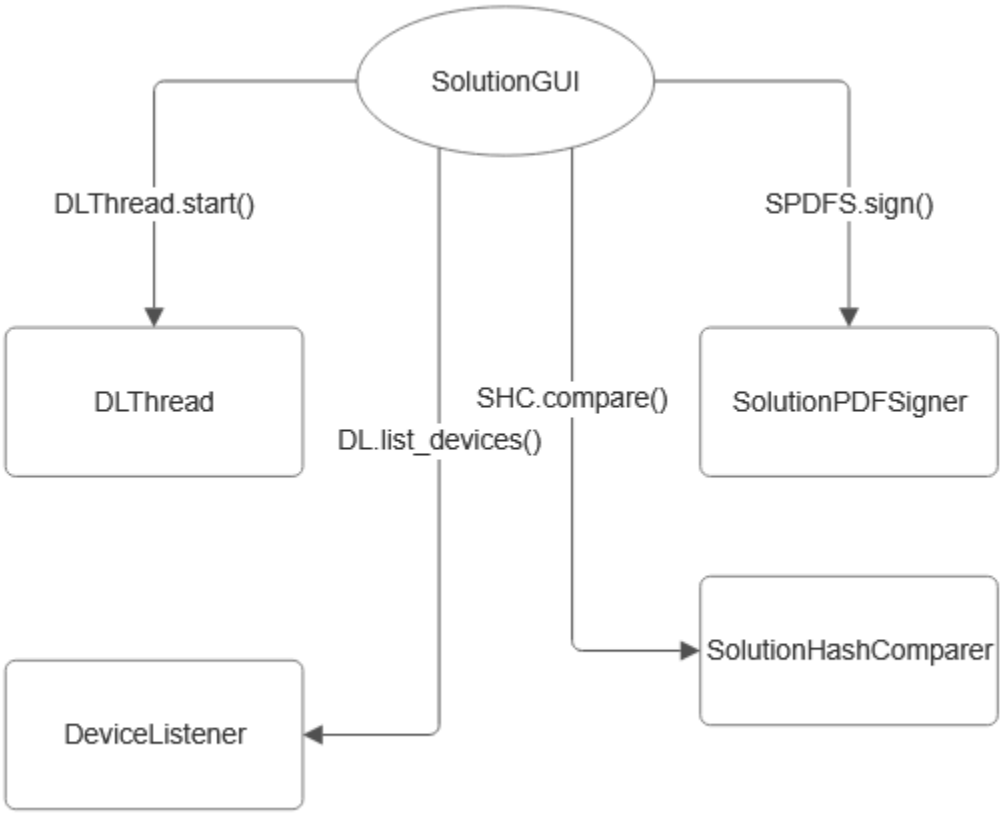

Fig 7. Simplified block diagram of the Solution

Explanation:

The Solution will have a structure rather similar to the AuxiliaryApp. The SolutionGUI, DLThread, and DeviceListener will have the same, or highly similar, purpose as their auxiliary counterparts.

The SolutionPDFSigner will be invoked by the SolutionGUI class to sign the chosen .pdf document on the user's demand. The resulting hash will be available for verification by the SolutionHashComparer class.