**Security of Computer Systems**

**Project Report**

Authors:

Tomasz, Sankowski, 193363

Piotr, Sulewski, 192594

GitHub repository:

<https://github.com/peterprospl12/pades-pdf-signer>

Version: 1.1

**Versions**

|  |  |  |
| --- | --- | --- |
| Version | Date | Description of changes |
| 1.0 | 01.04.2025 | Creation of the document |
| 1.1 | 03.04.2025 | Add “Testing” description |
| 1.2 |  |  |

1. **Project**
   1. ***Description***

The goal of this project is to develop an application that emulates a qualified electronic signature according to a simplified PAdES standard. The application allows users to sign PDF documents using a 4096-bit RSA private key, which is securely stored on a USB drive. When generating a new private key, the user sets a PIN, which is then hashed using SHA-256 along with a randomly generated salt to enhance security. The resulting hash is used to encrypt the private key with the AES-256 algorithm. Additionally, a randomly generated initialization vector (IV) is used during encryption, further strengthening security. Both the salt and the IV are stored next to the encrypted private key to ensure proper decryption.

To sign a document, the user provides the correct PIN, which is hashed again using the stored salt, allowing the application to reconstruct the encryption key. This key is then used along with the stored IV to decrypt the private key. If the provided PIN is incorrect, the resulting key will be invalid, making decryption unsuccessful. The application automatically detects USB drives and checks for the presence of a private key. Once the correct PIN is entered, the user can select any PDF file on their computer and sign it.

The signing process involves computing the hash of the PDF file (excluding metadata) and embedding the hash, along with the user’s name and signing date, into the new PDF metadata. This creates a signed duplicate of the original file. When generating a private key, the user also selects a storage location for the corresponding public key. The public key is used within the application to verify the PDF signature. During verification, the application recomputes the PDF hash and decrypts the signed hash from the document using the sender’s public key. If the decrypted hash matches the newly computed one, the signature is confirmed as valid.

Additionally, the project includes documentation generated with Doxygen and a GitHub repository containing the source code and commit history.

Obraz zawierający tekst, Czcionka, numer, linia

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

*Screen of running application.*

* 1. ***Diagram of the application***

Obraz zawierający tekst, zrzut ekranu, diagram, Prostokąt

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

*Fig. 1 – Block diagram.*

* 1. ***Crucial code fragments***
     1. ***Generating private and public key pair:***

***Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.***

* + 1. ***Converting private key into PEM format:***

***Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.***

* + 1. ***Generating random salt and using it to hash user’s PIN***

***Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.***

* + 1. ***Generating random initialization vector and using it and hashed PIN to encrypt private key:***

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

* + 1. ***Decrypting private key (hashing PIN using salt and then decrypting encrypted key). If wrong PIN is provided, decryption will create invalid PEM key which will throw an exception:***

***Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.***

* + 1. ***Hashing PDF without metadata and signing the hash with decrypted private key:***

***Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.***

* + 1. ***Add metadata with signature, signer name and signing date to new, signed PDF:***

***Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.***

* + 1. ***Hashing signed PDF file without metadata:***

***Obraz zawierający tekst, Czcionka, zrzut ekranu

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.***

* + 1. ***Comparing hash of PDF without metadata with hash in signed PDF metadata after decrypting it using public key:***

***Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.***

* 1. ***Summary***

The project’s code is available on GitHub repository under following link:

* <https://github.com/peterprospl12/breathing-classification-v2>

Repository is under MIT license which means it can be freely used, modified and distributed as long as the original copyright and license notice is kept.

Besides code, repository consists of documentation created using Doxygen tool. In root directory there is Doxyfile which let us automatically create documentation. Created documentation is in ./docs file. Repositorium has README file which has most of the necessary information about the project and running it.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

*Screen of documentation created using Doxygen.*

1. **Testing**
   1. **Application launch**
      1. Navigate to the application directory
      2. Run “python run\_pades\_signer.py”
      3. Verify the main window appears with three tabs: Sign, Verify, and Key Management
   2. **Verify key pair generation functionality**
      1. Insert a USB drive into the computer
      2. Open the application and navigate to the Generate Key tab
      3. Wait for the USB drive to be detected (check status indicator)
      4. Enter a PIN in the PIN input field
      5. Select a directory for public key storage
      6. Click “Generate and Save Key”
      7. Verify the status indicates success
      8. Check that a “public\_key.pem” file exists in the chosen directory
      9. Check that a “private\_key.key” file exists on the USB drive
   3. **Verify that the private key is properly protected with a PIN**
      1. Complete Test 2 to generate a key pair
      2. Navigate to the Sign Document tab
      3. Select a PDF file using the “Select PDF” button
      4. Slick “Sign Document”
      5. Enter an incorrect PIN when prompted
      6. Verify an error message is displayed
      7. Try again with the correct PIN
      8. Verify the document signing proceeds
   4. **Sign PDF Document**
      1. Insert USB drive with a previously generated private key
      2. Navigate to the Sign Document tab
      3. Select a PDF file using the “Select PDF” button
      4. Click “Sign Document”
      5. Enter the correct PIN when prompted
      6. Enter a signer name when prompted
      7. Verify the status indicates successful signing
      8. Check that a new file with “signed\_” prefix exists in the same directory as the original PDF
   5. **Verify the visual appearance of the signature in the document**
      1. Complete Test 4 to sign a document
      2. Open the signed PDF in a PDF viewer
      3. Confirm the PDF file metadata contains the signer name, signature and date information
   6. **Verify that a correctly signed document passes verification**
      1. Navigate to the Verify Signature tab
      2. Select the public key file using "Select Public Key" button
      3. Select a PDF that was signed in Test 4
      4. Click "Verify Signature"
      5. Verify that the status indicates "Signature is valid"
   7. **Verify that a tampered document fails verification**
      1. Take a signed PDF from Test 4
      2. Open it in a PDF editor
      3. Modify the content (add/underline/colour/remove text)
      4. Save the modified PDF
      5. In the application, navigate to the Verify tab
      6. Select the correct public key
      7. Select the modified PDF
      8. Click "Verify Signature"
      9. Verify that the status indicates the signature is invalid
   8. **Verify application behaviour when USB is removed unexpectedly**
      1. Insert USB drive with a private key
      2. Start the application and verify the USB is detected
      3. Navigate to the Sign Document tab and select a PDF
      4. Remove the USB drive without ejecting
      5. Click "Sign Document"
      6. Verify the application shows an appropriate error message
      7. Reinsert the USB drive
      8. Verify the application detects the USB drive again
   9. **Verify validation of user inputs.**
      1. Navigate to the Generate Key tab
      2. Click "Generate Key Pair" without entering a PIN
      3. Verify error message about missing PIN
      4. Enter a PIN but don't select a directory for public key
      5. Click "Generate Key Pair"
      6. Verify error message about missing public key path
2. **Literature**

[1] Cryptography module python documentation, <https://cryptography.io/en/latest/hazmat/primitives>

[2] Doxygen documentation, <https://www.doxygen.nl/manual/>

[3] PyQt5 (GUI library) documentation, <https://www.riverbankcomputing.com/static/Docs/PyQt5/>