Security of Computer Systems

Project Report

Authors:
Stefan, Furmański, 193090
Gracjan, Żukowski, 193184

Version: 1.0

Versions

Version	Date	Description of changes
1.0	14.03.2025	Creation of the document

1. Project – control and final term

1.1 Description

The project's goal was to use the PAdES (PDF Advanced Electronic Signature) standard to create a software application that would simulate a qualified electronic signature. Designing and creating a primary application for signing and confirming PDF documents as well as a secondary application for creating and protecting RSA keys were the main goals.

1.2 Results

RSA Key Generation and Storage

- Developed an auxiliary application for generating RSA key pairs.
- Implemented encryption of the private key using AES-256 with a PIN-derived key.
- Enabled secure storage of the encrypted private key on a USB drive.

Automatic USB Drive Detection

- Added automatic detection of the USB drive containing the encrypted RSA key.

Document Signing and Signature Embedding

- Implemented the PAdES standard for adding the digital signature directly into the PDF document (meta data).

Signature Verification

- Designed a process that uses the public key to verify signed document.
- Implemented resistance to document tampering by validating document integrity.

Graphical User Interface (GUI)

- Created interface for main and auxiliary applications.

Testing and Validation

- Verified correct and incorrect signature validation scenarios.
- Tested encryption and decryption of the private key and document signatures.

Documentation and Code Repository

- Generated full PDF code documentation using Doxygen.
- Code stored in Github repository.

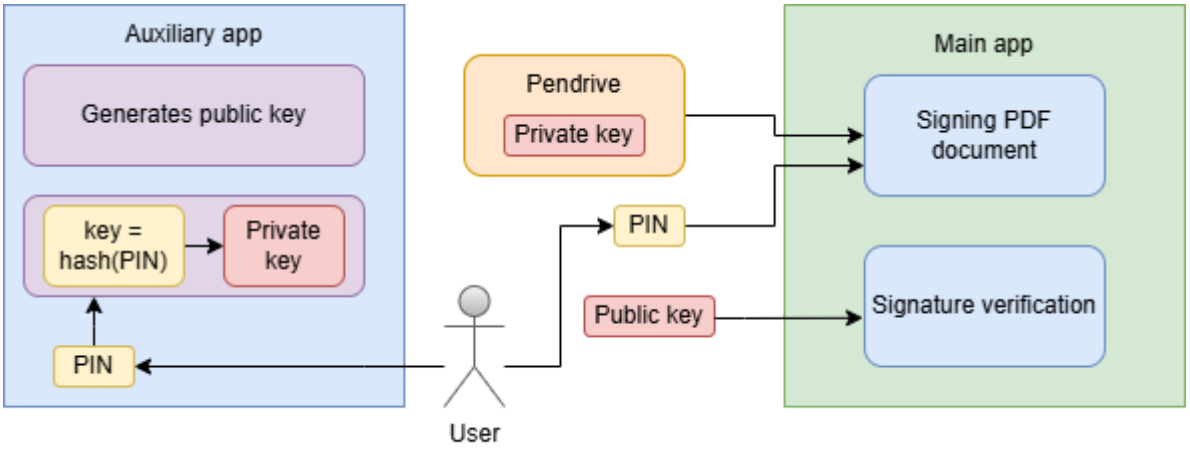

Fig. 1 – Block diagram of the project concept.

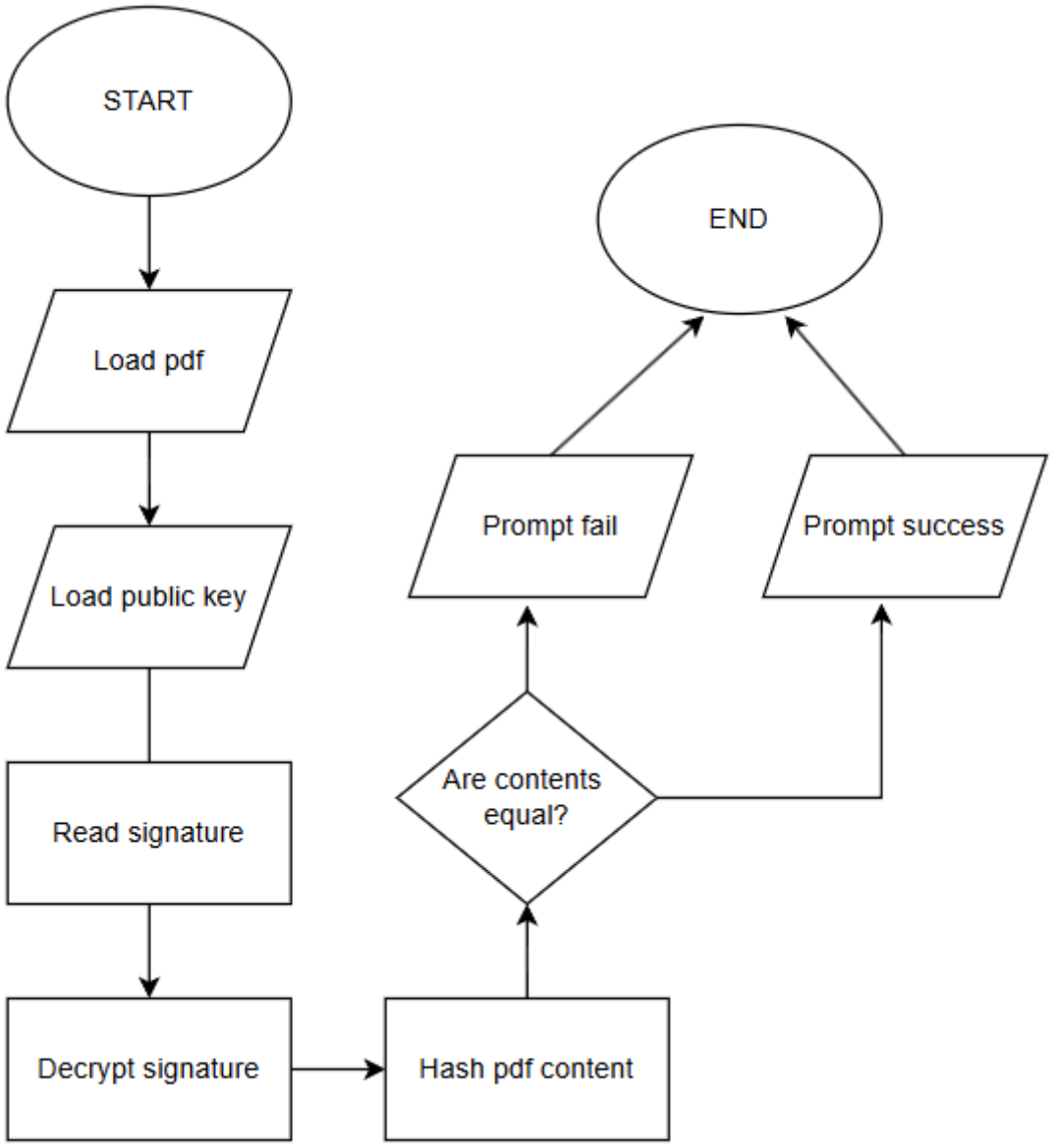

Fig. 2 – Flow diagram of signing process.

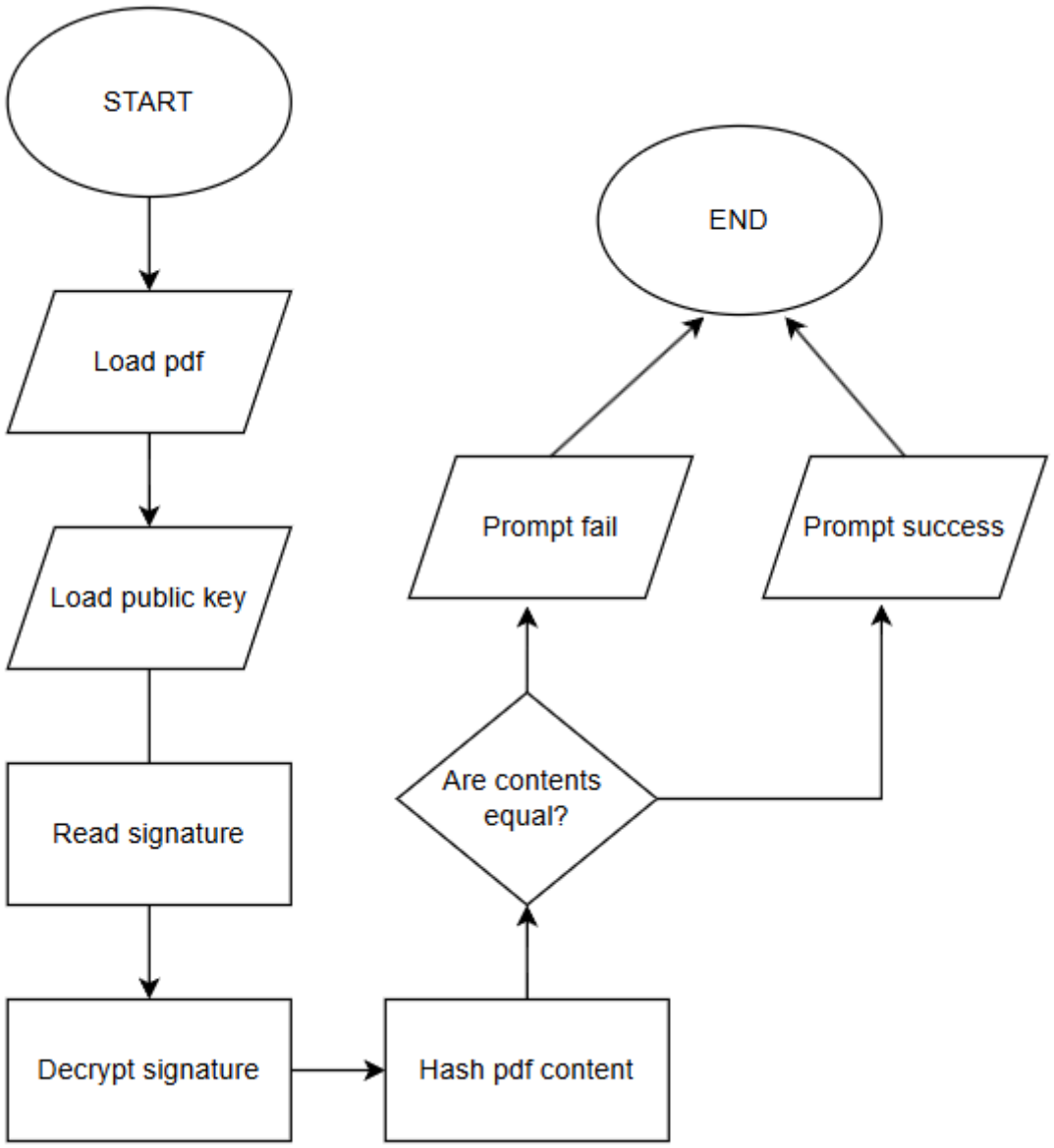

Fig. 3 – Flow diagram of verifying process.

1.3 Code Description

Auxiliary Application: RSA Key Generation

The auxiliary application provides functionality for generating RSA keys and securely storing the private key.

Key Components:

- *KeyGeneratorWindow* (Located in auxiliary_app/gui/key_generator_window.py)

open_pin_pad(): Opens a dialog for the user to input a PIN.

start_key_generation(pin): Begins the key generation process and encrypts the private key using AES-256.

handle_status(status_code, message): Displays the progress and handles success or failure messages.

- *KeyGenerationThread* (Located in auxiliary_app/gui/key_generation_thread.py)

run(): Performs RSA key generation and encryption in a separate thread for improved responsiveness.

Main Application: Document Signing and Verification

The main application handles PDF signing and verifying using the RSA key stored on the USB drive.

Key Components:

- *SignVerifyWindow* (Located in main_app/gui/sign_and_verify.py)

sign_pdf(): Initiates the signing process by loading the private key from the USB drive and embedding the signature.

```
def sign_pdf(pdf_path: str, rsa_key: RSA.RsaKey, progress_signal=None):
check_pdf_exists(pdf_path, progress_signal)
    try:
     pdf_path = initialize_signing_process(pdf_path, progress_signal)
        pdf_content = read_pdf_file(pdf_path)
        pdf_hash = hash_pdf(pdf_content, progress_signal)

        temp_pdf_path = clear_signature_metadata(pdf_path)
        pdf_content = read_pdf_file(temp_pdf_path)
        pdf_hash = hash_pdf(pdf_content, progress_signal)

        signature = create_signature(rsa_key, pdf_hash, progress_signal)
        result_path = add_signature_to_pdf(temp_pdf_path, signature,
progress_signal)
        pdf_content = read_pdf_file(result_path)
        pdf_hash = hash_pdf(pdf_content, progress_signal)
    except Exception:
        logger.exception("Error while signing PDF File: %s")
        raise
```

verify_sign(): Starts the verification process by comparing the embedded signature against a provided public key.

```
def verify_pdf(pdf_path: str, public_key: RSA.RsaKey, progress_signal=None) -> bool:

check_pdf_exists(pdf_path, progress_signal)
    try:
        reader, signature = read_pdf_metadata(pdf_path, progress_signal)
        pdf_hash = prepare_unsigned_pdf(reader, pdf_path, progress_signal)
        verify_signature(public_key, pdf_hash, signature, pdf_path, progress_signal)
    except Exception:
        logger.exception("Error verifying signature: %s", pdf_path)
        raise
```

- *SignThread* (Located in main_app/gui/sign_thread.py)

run(): Executes the PDF signing in a separate thread, providing real-time updates on the progress.

- *VerifyThread* (Located in main_app/gui/verify_thread.py)

run(): Handles the signature verification, validating document integrity.

Encryption and Decryption Utilities

Key encryption and decryption processes are handled by the crypto_utils.py module:

- *generate_rsa_keys(pin, drive_manager)* (Located in auxiliary_app/utils/utils.py)

  Generates RSA key pairs and encrypts the private key using the user-provided PIN.

- *decrypt_rsa_key(pin, drive_manager)* (Located in main_app/utils/crypto_utils.py)

  Decrypts the private RSA key using the PIN and the USB drive.

- *read_public_key(public_key_path)* (Located in main_app/utils/crypto_utils.py)

  Loads the public key from the specified file for signature verification.

Drive Management

Drive detection and file storage are handled by the DriveManager class (Located in common/drive_manager/drive_manager.py):

- *refresh()*: Detects connected USB drives.
- *list_drives_with_keys()*: Identifies drives containing specific key files.
- *save_to_drive(data, destination_name)*: Stores encrypted key files on a selected USB drive.

1.4 Summary

The project's objective of putting in place a reliable and secure instrument to mimic the PAdES qualified electronic signature process was accomplished. The system's user-friendly GUI and extensive error handling allow for safe key generation, document signing, and verification. Every feature complies with the standards, including the integration of USB storage devices, the usage of RSA-4096 keys, and AES-256 encryption.

2. Literature

[1] PyCryptodome's documentation, https://pycryptodome.readthedocs.io/en/latest/

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

[3] Qt for Python, https://doc.qt.io/qtforpython-6/

[4] PyPDF2, https://pypdf2.readthedocs.io/en/3.x/

[5] Digital signature theory, https://en.wikipedia.org/wiki/Digital_signature