PAdES

Security of Computer Systems

Project Report

Authors:
Krzysztof Nasuta, 193328
Filip Dawidowski, 193433

Version: 1.0

Versions

Version	Date	Description of changes
1.0	02.04.2025	Creation of the document

1. Project

1.1 Description

The primary objective of this project is to develop a software tool that emulates the process of creating a qualified electronic signature for PDF documents, adhering to the PAdES (PDF Advanced Electronic Signature) standard. The project involves designing a main application for signing PDFs and an auxiliary application for generating and securing RSA key pairs.

1.2 Results

The project was implemented with Python programming language. It consists of two main applications - one for generating the RSA key pair and securely storing the private key on the USB drive, and second one used for signing the PDF files and verifying the signatures.
Each application has its own startup Python file: generate_signing_key.py for the auxiliary application, and pades.py for the main application.
Common code related to GUI was placed in the windows package, and general code implementing application functionality is stored in the lib package.

Application source code: github.com/Nasus20202/pades

General flow of the application:

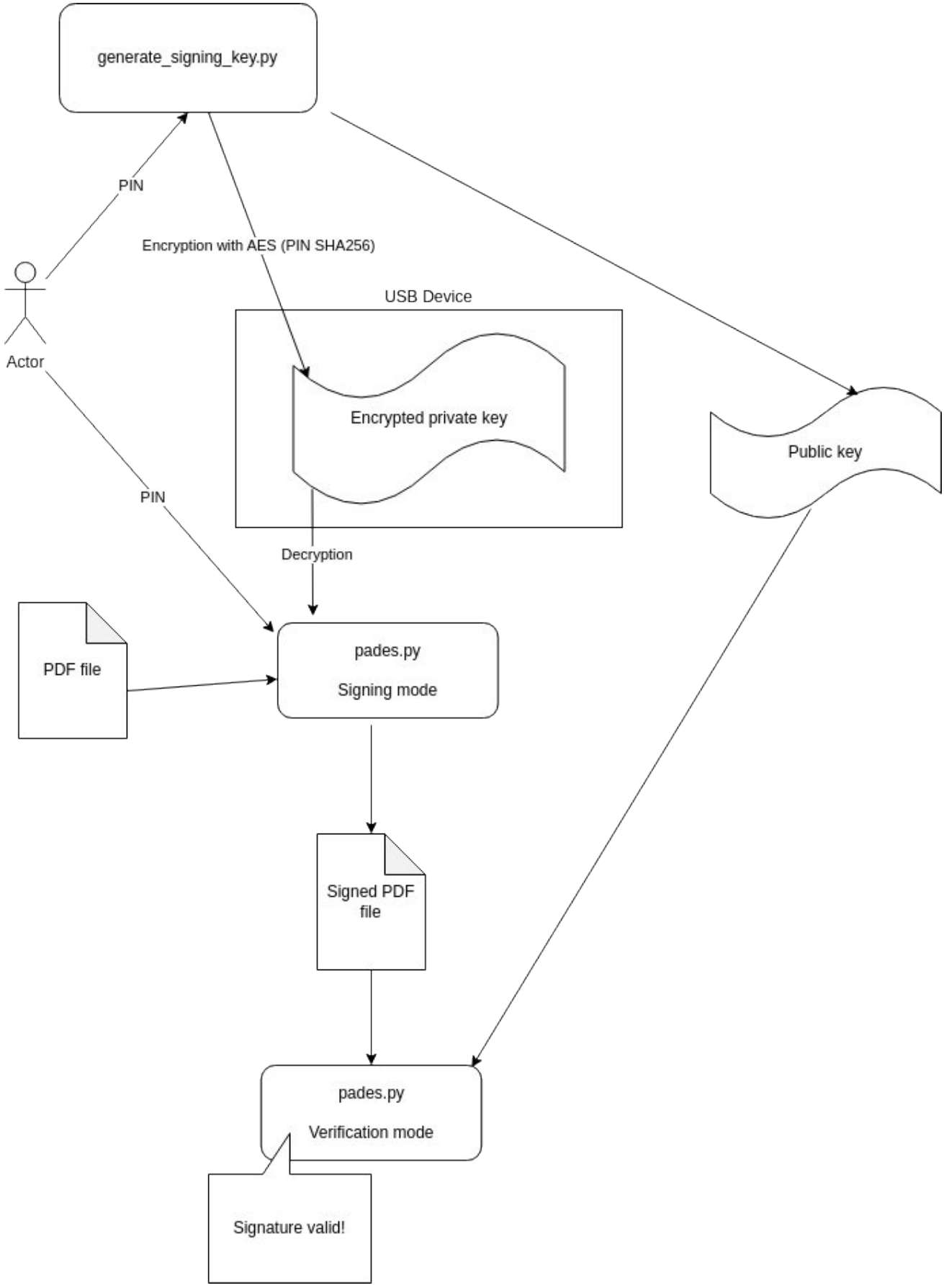


Fig. 1 – Flow diagram.

1.3 How to use the application?

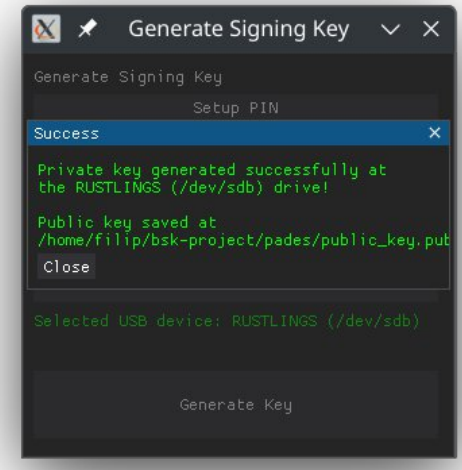
1.3.1 Generate Signing Key App

First, you need to generate a pair of RSA keys. To do this, you need to open the generate_signing_key.py app. Here is the main window of this application:

Now, you need to insert the USB drive to the computer and mount it in the system. The USB drive will be shown in „Select USB drive” section (in the example above the USB drive RUSTLINGS is shown). Select it by clicking.

Click on „Setup PIN” button to provide the code which would be used to encrypt the signing key. Enter your desired PIN and click „Submit”.

Now, click on „Generate Key”. The keys will be generated and the success window would appear:

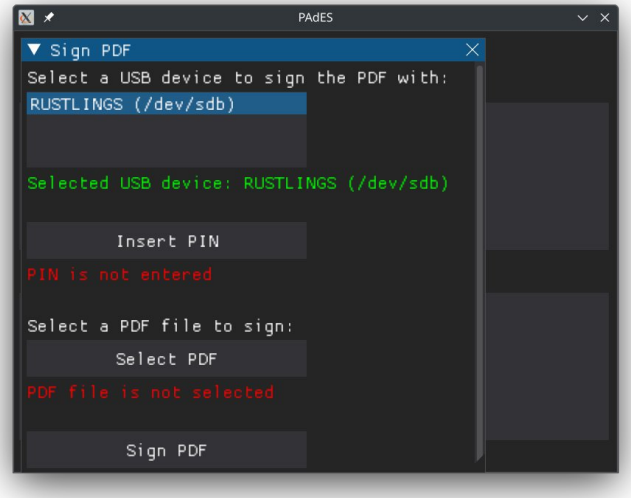


1.3.2 PAdES app

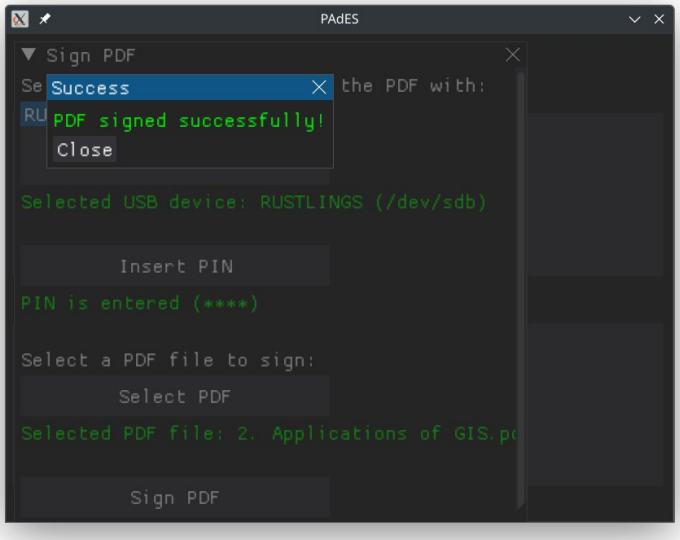
In order to sign or verify the PDFs use the pades.py application. Here is the main window of it:

To begin, insert the USB drive with previously generated private key to the computer and mount it in the system.

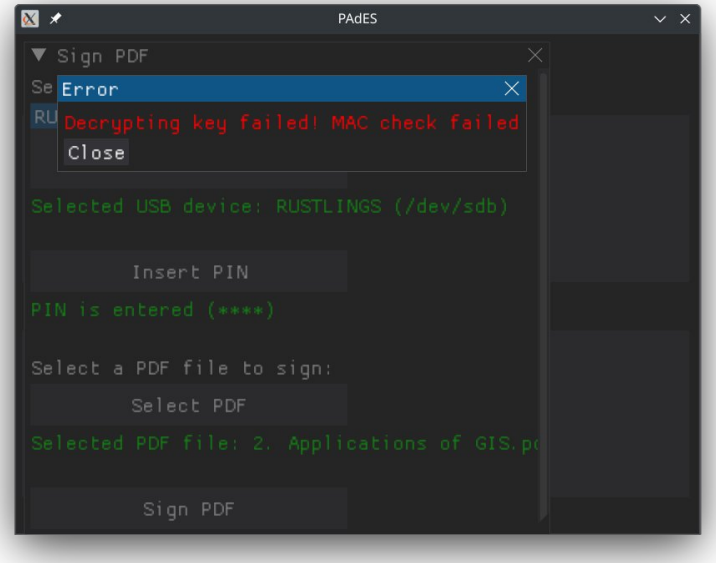
To sign a PDF, click on „Sign PDF” button. The following subwindow would appear:



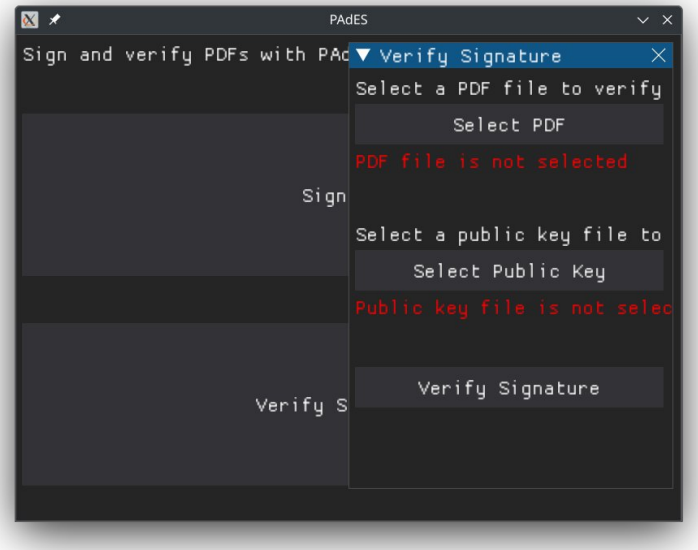
First, select the USB drive by clicking on it. Then click on „Insert PIN” and provide your pin. By clicking the „Select PDF” button you can select the PDF file which you would like to sign. When all is set, click on „Sign PDF”. When all provided information is correct, you should see the success confirmation:



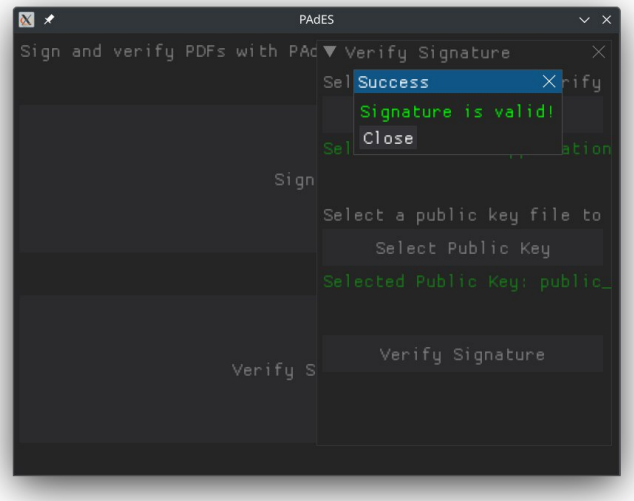
The signed PDF would be saved as a copy of the original one, with suffix „_signed.pdf”. In case the wrong pin is provided, the signed file would not be generated and the appropriate information would be shown:



To verify the signed PDF click on „Verify Signature” in the main app window. The following subwindow would appear:



First, select the PDF whose signature you want to verify. After that, click on „Select Public Key” and select the „.pub” file containing the public key which would be used to verification of the signature. By default, it is located in the app folder. When all is set, click on „Verify Signature”. If the selected public key is correct and the signed PDF was not modified, you will see the success confirmation like this:



Whereas when something is wrong, the error message would appear.

1.4 Code Description

General code documentation was created with Doxygen. It's accessible in the project Github repository. Documentation can be generated with the doxygen cli tool.

Code calculating the SHA256 hash used for AES encrypting with PIN

```
def hash_string(string: str) -> str:
    """!
    Hash a string using SHA256.
    @param string: The string to hash.
    @return The hashed string."""
    return SHA256.new(string.encode("utf-8")).digest()
```

Generating the RSA key pair:

```
def generate_rsa_key_pair(key_size: int = 4096) -> tuple[bytes, bytes]:
    """!
    Generate a RSA key pair.
    @param key_size: The size of the key in bits.
    @return A tuple containing the private and public key."""
    key = RSA.generate(key_size)
    private_key = key.export_key()
    public_key = key.publickey().export_key()
    return private_key, public_key
```

AES encryption:

```
def encrypt_data_with_aes(data: bytes, key: bytes) -> bytes:
    """!
    Encrypt data using AES.
    @param data: The data to encrypt.
    @param key: The key to use for encryption.
    @return The encrypted data."""
    cipher = AES.new(key, AES.MODE_EAX)
    ciphertext, tag = cipher.encrypt_and_digest(data)
    return cipher.nonce, tag, ciphertext
```

AES decryption:

```
def decrypt_data_with_aes(
    nonce: bytes, tag: bytes, ciphertext: bytes, key: bytes
) -> bytes:
    """!
    Decrypt data using AES.
    @param nonce: The nonce used for encryption.
```

```
    @param tag: The tag used for encryption.
    @param ciphertext: The encrypted data.
    @param key: The key to use for decryption.
    @return The decrypted data."""
    cipher = AES.new(key, AES.MODE_EAX, nonce=nonce)
    return cipher.decrypt_and_verify(ciphertext, tag)
```

Signing the PDF file

```
def sign_pdf(file_path: str, private_key: bytes, signed_file_path=None)
-> None:
    """!
    Sign a PDF file with a private key and save it to the signed_file_path.
    The signature is constructed by hashing the PDF file and then
signing the hash with the provided private key.
    This signed hash is then appended to the end of the file.
    @param file_path: The path to the PDF file to sign.
    @param private_key: The private key to sign the PDF file with.
    @param signed_file_path: The path to save the signed PDF file to.
    If None, the file will be saved in the same directory with the same
name but with "_signed" suffix.
    @return None
    """
    if signed_file_path is None:
        signed_file_path = file_path.replace(".pdf", "_signed.pdf")

    with open(file_path, "rb") as f:
        data = f.read()

    rsa_key = RSA.import_key(private_key)
    signature = pss.new(rsa_key).sign(SHA256.new(data))

    with open(signed_file_path, "wb") as f:
        f.write(data)
        f.write(signature)
```

PDF file signature verification

```
## @var SIGNATURE_LENGTH
# The length of the signature in bytes.
SIGNATURE_LENGTH = 512

def verify_pdf(file_path: str, public_key: bytes) -> bool:
```

```
    """!
    Verify the signature of a signed PDF file.
    The signature is verified by hashing the PDF contents (excluding
embedded signature) and then comparing the hash
    to the signature extracted from the file (the signature is first
decrypted with the public key).
    @param file_path: The path to the signed PDF file to verify.
    @param public_key: The public key used to verify the signature.
    @return True if the signature is valid, False otherwise.
    """
    with open(file_path, "rb") as f:
        data = f.read()

    pdf_data = data[:-SIGNATURE_LENGTH]
    signature = data[-SIGNATURE_LENGTH:]

    rsa_key = RSA.import_key(public_key)
    pdf_hash = SHA256.new(pdf_data)
    verifier = pss.new(rsa_key)
    try:
        verifier.verify(pdf_hash, signature)
        return True
    except (ValueError, TypeError):
        return False
```

1.5 Github repository

Source code of the application is stored in this Github repository: Nasus20202/pades.

2. Literature

[1] https://en.wikipedia.org/wiki/PAdES

[2] https://github.com/hoffstadt/DearPyGui

[3] https://www.doxygen.nl/

[4] https://www.pycryptodome.org/

[5] Project instruction