Security of Computer Systems

Project Report

Authors:
Patryk Sowiński-Toczek, 191711
Arkadiusz Szamocki,

Version: 1.0

Versions

Version	Date	Description of changes
1.0	13.04.2025	Powstanie dokumentu

1. Project – control term

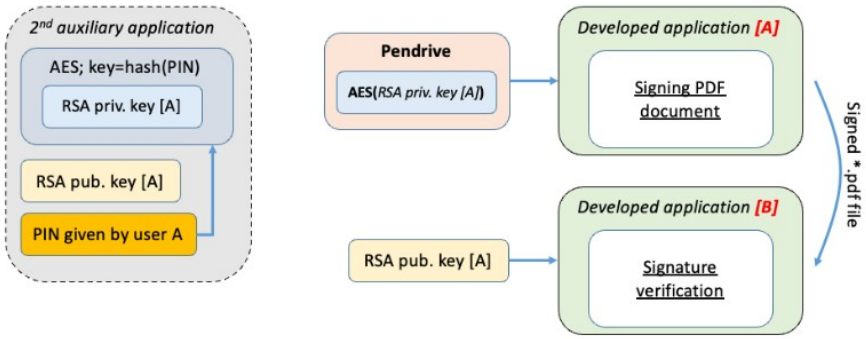
1.1 Description

W ramach zajęć projektowych koniecznym jest przygotowanie dwóch aplikacji - głównej oraz dodatkowej, którę będą umożliwiały użytkownikowi wygenerowanie pary klucza prywatnego oraz klucza publicznego (aplikacja dodatkowa), które następnie posłużą do podpisywania oraz weryfikowania plików w formacie PDF zgodnie ze standardem PAdES.

Do realizacji projektu został wykorzystany język Python wraz z interfejsem Tk (tkinter) oraz bibliotekami pycryptodome oraz cryptography.

Założenia projektu są wyszczególnione w osobnym pliku znajdującym się na platformie eNauczanie w kursie "Bezpieczeństwo Systemów Komputerowych - 2025".

Schemat docelowego projektu:



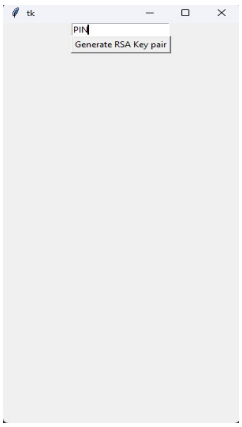
Rysunek 1 - Schemat blokowy konceptu projektu do zrealizowania

1.2 Results

Na termin kontrolny została wykonana aplikacja dodatkowa umożliwiająca generowanie pary kluczy publicznego i prywatnego z wykorzystaniem bibliotek pycryptodome oraz cryptography. Klucz prywatny został zabezpieczony z wykorzystaniem algorytmu szyfrującego AES w trybie ECB oraz algorytmu haszującego SHA-256. Interfejs aplikacji powstał w oparciu o Tkinter.

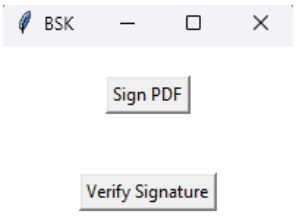
Użytkownik podaje 16 bajtowy PIN (długość jest walidowana przez aplikację), który jest przetwarzany przez funkcję skrótu SHA-256 tworząc 32 bajtowy klucz szyfrujący, przy wykorzystaniu którego klucz prywatny jest szyfrowany AES w trybie ECB.

Na dysku, w lokalizacji w której przechowywana jest aplikacja, powstają dwa pliki - encrypted - z zaszyfrowanym kluczem prywatnym, oraz encrypted.pub - z kluczem publicznym.

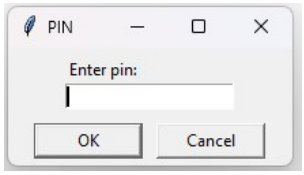


Rysunek 2 - Interfejs aplikacji dodatkowej

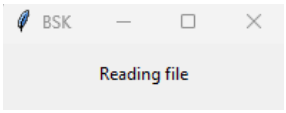
Powstał również szkielet aplikacji głównej, który umożliwia odszyfrowanie klucza prywatnego powstałego w aplikacji dodatkowej po podaniu PINu. Interfejs aplikacji powstał w oparciu o Tkinter. Przyciski "Sign PDF" oraz "Verify Signature" umożliwiają wskazania pliku w formacie pdf. W przypadku "Verify Signature", aplikacja informuje o swoim stanie (tj. wyświetla informację o odczytywaniu pliku).



Rysunek 3 - Ekran główny aplikacji



Rysunek 4 - Okno z prośba o wpisanie PINu użytkownika



Rysunek 5 - Okno informujące użytkownika o stanie aplikacji

1.3 Summary

W ramach terminu kontrolnego zajęć projektowych powstał szkielet aplikacji, która docelowo będzie służyła do podpisywania plików w formacie pdf kluczem prywatnym zgodnie ze standardem PAdES oraz weryfikowania podpisu z wykorzystaniem klucza publicznego.

Pliki źródłowe są dostępne na platformie GitHub pod adresem: https://github.com/Shubale/pdfsinger

2. Literature

[1] https://ec.europa.eu/digital-building-blocks/DSS/webapp-demo/doc/dss-documentation.html#_pades_signature_pdf

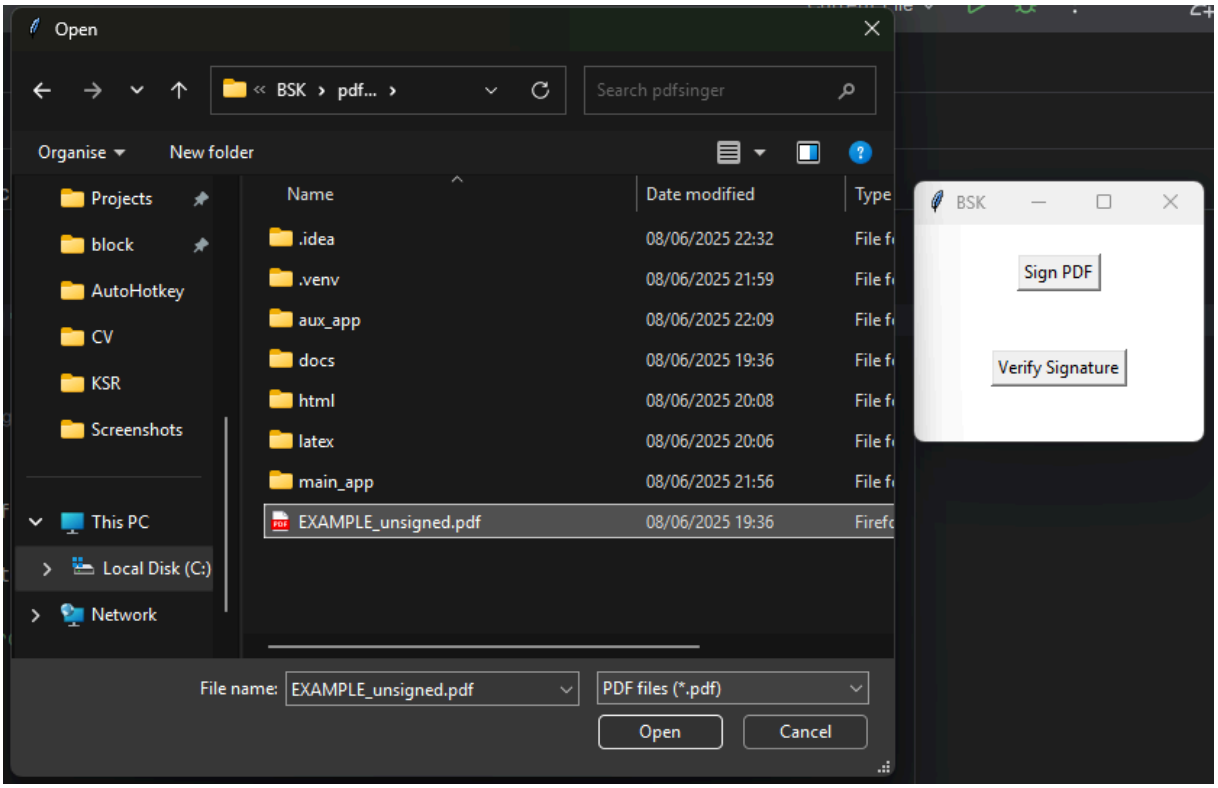
[2] https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/PDF32000_2008.pdf

[3] https://www.etsi.org/deliver/etsi_en/319100_319199/31914201/01.02.01_60/en_31914201v010201p.pdf

2. Project – Final term

2.1 Key requirements:

- **The GUI interface must allow to select any document (*.pdf) that will be signed:**

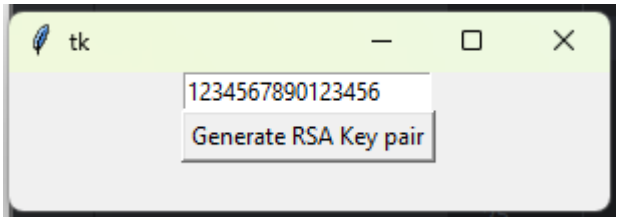


- **The signature must use the RSA algorithm with a 4096-length key:**

```
def generate_rsa_pair(s):
    if not validate_pin(s):
            print('You PIN number has to have 16 bytes; currently ' +
str(len(s.encode('utf-8'))))
        return
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=4096
    )
```

- **A pseudorandom generator must be used to generate the RSA keys:**
  The above mentioned code contains rsa.generate_private_key() function which utilizes Pseudo Random Number Generation[1]
- **The private key stored on the pendrive must be encrypted by the AES algorithm, where the 256-bit key is the hash from PIN known only to user A:**

User is prompted to create PIN password that has to have exactly 16 bytes (for security measure; comparing low-character hashes are trivial to break)



This PIN code is then hashed using SHA256 and then encrypted using AES with ECB mode (ECB was used for simplicity of its implementation):

```
digest = hashes.Hash(hashes.SHA256())
digest.update(s.encode('utf-8'))
key = digest.finalize()

cipher = AES.new(key, AES.MODE_ECB)
ciphertext = cipher.encrypt(pad(pem, BLOCK_SIZE))
```

- **Pendrive usage for storing the private RSA key is obligatory. The pendrive must be detected and the encrypted RSA key automatically loaded to the main application.**
  The key is then decrypted and then used by application to sign the PDF

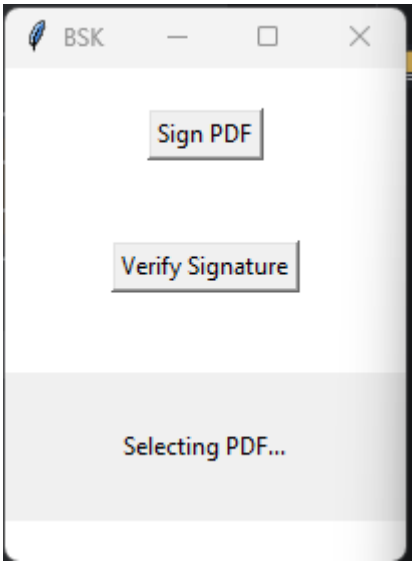
```
def read_private_key(self, pin):
    usb_path = "G:\\encrypted"
    try:
        with open(usb_path, "rb") as f:
            key = f.read()
            return self.decrypt_private_key(pin, key)
    except:
            messagebox.showerror("Error", "Failed to read private key
file")
        return None
```

- **The public key can be stored on the hard disk of the computer or be transferred to another physical computer to verify the signature:**
  RSA private and public keys are stored as files, so they can be freely used in any digital and analog storage

```
f = open("encrypted", "w")
f.write(str(ciphertext))
f.close()
f = open("encrypted.pub", "w")
f.write(str(public_key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
)))
f.close()
```

- **It is obligatory to implement status/message icons to present the state of the application (recognition of hardware tool, reading the private key, signature status).**

This text will be updated through out the application during the usage of the application, for example while selecting a PDF:




```
self.status = tk.Label(self.mainframe, text='No PDF selected',
height=10, width=40)
self.status.pack(pady=20)
```

.
.
.

```
self.status.config(text='Reading private key...')
```

2.2 Realized tasks

1. Implemented auxiliary application
    a. used python for code logic, tkinter for interface
    b. cryptographic libraries used are pycryptodome and cryptography
    c. the app prompts user to type in 16-byte PIN

d. this PIN is then hashed using SHA256 algorithm
e. using that hash RSA key pair is generated (keysize=4096)
f. key pair is then exported to 2 files, encrypted (private key) and encrypted.pub (public key)
g. private key is to be stored on a pendrive

2. Implemented main app
   a. used python for code logic, tkinter for interface
   b. cryptographic libraries used are pycryptodome and cryptography
   c. in normal usage user is prompted to first sign a PDF
   d. pdf signing is separated into 6 steps:
      i. selecting PDF using file system (filedialog from tkinter)
      ii. reading private key from pendrive (G:\encrypted)
      iii. decrypting said key using PIN, thus receiving decrypted private key
      iv. hashing PDF (SHA256)
      v. creating signature using private_key.sign()

```
signature = private_key.sign(
    hash_bytes,
    padding.PKCS1v15(),
    hashes.SHA256()
)
```

      vi. appending signature to PDF
   e. after doing that user can verify given signature in following 6 steps:
      i. selecting PDF using file system (filedialog from tkinter)
      ii. reading public key (encrypted.pub file)
      iii. determining signature length

```
SIGNATURE_LENGTH = pub_key.key_size // 8
```

      iv. PDF is separated into data and signature
      v. PDF data is hashed (SHA256, as earlier)
      vi. signature is verified based on hashed data, signature and public

```
pub_key.verify(
    signature,
    expected_hash,
    padding.PKCS1v15(),
    hashes.SHA256()
)
```

2.3 Github repository

https://github.com/Shubale/pdfsinger

1. Literature

[1] https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/

[2] https://ec.europa.eu/digital-building-blocks/DSS/webapp-demo/doc/dss-documentation.html#_pades_signature_pdf

[3] https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/PDF32000_2008.pdf

[4] https://www.etsi.org/deliver/etsi_en/319100_319199/31914201/01.02.01_60/en_31914201v010201p.pdf