Security of Computer Systems

Project Report

Authors:
Maciej, Raciniewski, 189774
Damian, Trowski, 193443

Version: 1.0

Versions

Version	Date	Description of changes
1.0	09.04.2025	Creation of the document
1.1	10.06.2025	Completion of the document

- ## Project – control term
- ***Description***

Projekt składa się z dwóch aplikacji stworzonych przy użyciu frameworka Electron:

-Aplikacja pomocnicza (Auxiliary Application) – zajmuje się generowaniem kluczy RSA oraz ich zabezpieczaniem. Logika zaimplementowana jest w JavaScript.

- Główna aplikacja (Main Application) – umożliwia deszyfrowanie klucza RSA i pracę z podpisami cyfrowymi, w tym weryfikację podpisów dokumentów. Logika zaimplementowana jest w Pythonie.

- ***Results***

Funkcjonalności aplikacji pomocniczej (Auxiliary Application)

1. Generowanie pary kluczy RSA

-Aplikacja tworzy parę kluczy RSA: klucz publiczny oraz klucz prywatny.

-Klucz publiczny służy do weryfikacji podpisów, a klucz prywatny do ich generowania.

2. Szyfrowanie klucza prywatnego

-Klucz prywatny jest zabezpieczany poprzez szyfrowanie algorytmem AES.

-Do zaszyfrowania klucza prywatnego używany jest PIN podany przez użytkownika.

Funkcjonalności głównej aplikacji (Main Application)

1. Deszyfrowanie klucza RSA

-Użytkownik podaje PIN w aplikacji, aby odszyfrować klucz prywatny.

-Algorytm AES umożliwia przywrócenie klucza prywatnego na podstawie zaszyfrowanej wersji i podanego PIN-u.

2. Wykrywanie nośnika USB (pendrive)

-Aplikacja monitoruje system i automatycznie wykrywa, gdy użytkownik włoży pendrive.

3. Podpisywanie dokumentów .pdf

-Klucz prywatny (odszyfrowany z użyciem PIN-u) jest wykorzystywany do cyfrowego podpisywania dokumentów.

-W dokumencie pdf jest tworzony obiekt (placeholder), który służy do przechowywania podpisu o stałej długości dla klucza 4096 bitów.

-Generowany jest podpis poprzez podpisywanie skrótu dokumentu (skrót nie obejmuje placeholdera, jest obliczany algorytmem sha256) kluczem prywatnym.

4.Weryfikacja podpisów dokumentów

-Obliczany jest skrót pdfa za pomocą algorytmu sha256 (nie obejmujący placeholdera).

-Weryfikowany jest podpis za pomocą klucza publicznego oraz skrótu.

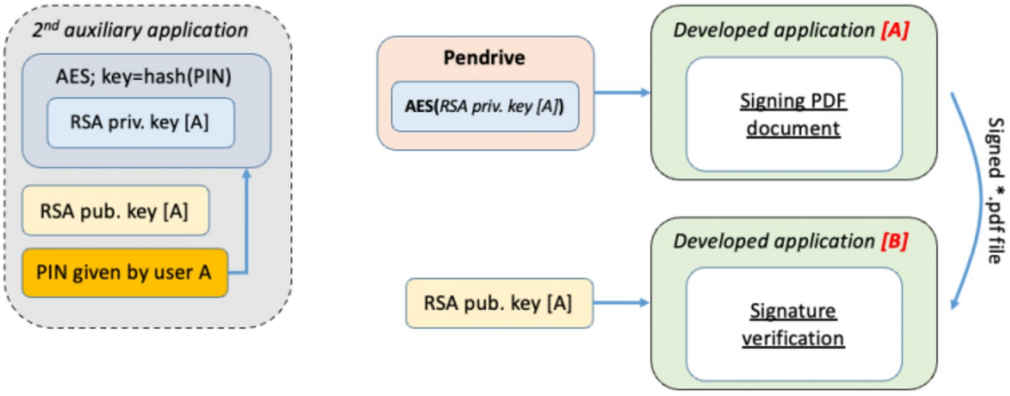


Fig. 1 – Block diagram of the project concept.

- ***Summary***

Do zrobienia zostało nam dopracowanie Frontendu dla głównej aplikacji oraz integracja poszczególnych komponentów aplikacji.

• Project – Final term

- ***Description***

Po terminie kontrolnym udało się zintegrować aplikacje podpisującą i weryfikującą podpis, została dodana możliwość podpisywania wielu dokumentów oraz został dopracowany frontend aplikacji.

- ***Code Description***

```
function generateRSAKeys(diskPath, userPIN) {
    return new Promise((resolve, reject) => {
        try {
            console.log('generateRSAKeys called with:', { diskPath, userPIN });
            const keypair = forge.pki.rsa.generateKeyPair({ bits: 4096 });
            const publicKeyPem = forge.pki.publicKeyToPem(keypair.publicKey);
            const privateKeyPem = forge.pki.privateKeyToPem(keypair.privateKey);

            const aesKey = hashPIN(userPIN);

            const encrypted = encryptWithAES(privateKeyPem, aesKey);

            const publicKeyPath = path.join(diskPath, 'rsa_public_key.pem');
            const privateKeyPath = path.join(diskPath, 'rsa_private_key.pem');

            fs.writeFileSync(publicKeyPath, publicKeyPem);
            console.log("Type of encrypted", typeof data)
            if (!Buffer.isBuffer(encrypted)) {
                throw new Error('Encryption failed. "encrypted" is not a Buffer.');
            }
            fs.writeFileSync(privateKeyPath, encrypted);

            resolve({ publicKeyPath, privateKeyPath });
        } catch (error) {
            reject(error);
        }
    });
}
```

List. 1 – Code listing [2].

- ***Description***

  Funkcja generuje parę kluczy RSA (4096-bit), szyfruje klucz prywatny przy użyciu podanego przez użytkownika PIN-u i zapisuje oba klucze na dysku w folderze określonym przez parametr diskPath. Zwraca obiekt zawierający ścieżkę do pliku z kluczem publicznym (publicKeyPath) oraz ścieżkę do pliku z zaszyfrowanym kluczem prywatnym (privateKeyPath). PIN (userPIN) musi być niepustym ciągiem znaków. Jeśli generowanie, szyfrowanie lub zapisywanie plików się nie powiedzie, funkcja zgłasza błąd.

```
function encryptWithAES(data, aesKey) {
    console.log('Type of data:', typeof data);
    console.log('Type of aesKey:', typeof aesKey);
    if (!aesKey || aesKey.length !== 32) {
        throw new Error('Invalid AES key. Expected a 256-bit key.');
    }
    const cipher = crypto.createCipheriv('aes-256-ecb', aesKey, null);
    cipher.setAutoPadding(true);
    try {
        const encrypted = Buffer.concat([cipher.update(data, 'utf8'), cipher.final()]);
        console.log('Type of encrypted:', typeof encrypted);
        if (!Buffer.isBuffer(encrypted)) {
            throw new Error('Encryption failed. "encrypted" is not a Buffer.');
        }
        return encrypted;
    } catch (error) {
        console.error('Encryption error:', error);
        throw error;
    }
}
```

List. 2 – Code listing [2].

- ***Description***

-

```
def create_pdf_placeholder(input_pdf_path: str, output_pdf_path: str, signature_length: int) -> None:
    with open(input_pdf_path, 'rb') as input_file, open(output_pdf_path, 'wb') as output_file:
        reader = pypdf.PdfReader(input_file)
        writer = pypdf.PdfWriter()

        for page in reader.pages:
            writer.add_page(page)

        # Tworzymy obiekt podpisu /Sig – BEZ formularzy, bez Annots
        sig_dict = DictionaryObject()
        sig_dict.update({
            NameObject("/Type"): NameObject("/BskSignature"),
            NameObject("/Contents"): create_string_object("0"*signature_length), #ByteStringObject( r
        })
        writer._add_object(sig_dict)
        with open(output_pdf_path, "wb") as f_out:
            writer.write(f_out)
```

- *List. 3 – Code listing [2].*

- ***Description***

Funkcja create_pdf_placeholder dodaje pole na podpis cyfrowy do pliku PDF. Wczytuje plik PDF, kopiuje jego zawartość, dodaje pole na podpis o określonym rozmiarze (w bajtach), a następnie zapisuje zmodyfikowany plik w nowym miejscu.

```python
def find_signature(pdf_path: str) -> Optional[int]:
    with open(pdf_path, "rb") as f:
        pattern = b"/Type /BskSignature\n/Contents ("
        position = f.read().find(pattern)

        if position != -1:
            position+=len(pattern)
            print(f"Placeholder: {position}")
            return position
        else:
            print("Placeholder not found.")
    return None
```

- *List. 3 – Code listing [2].*

- ***Description***

Funkcja find_signature szuka pola podpisu cyfrowego w pliku PDF, identyfikowanego przez wzorzec /Type /BskSignature\n/Contents (. Jeśli placeholder zostanie znaleziony, funkcja zwraca pozycję (offset w bajtach), w której zaczyna się zawartość podpisu. W przeciwnym razie zwraca None.

```python
def sign_hash_with_private_key(hash_value: str, private_key_path: str) -> str:
    with open(private_key_path, 'rb') as key_file:
        private_key = serialization.load_pem_private_key(
            key_file.read(),
            password=None,
            backend=default_backend()
        )
    signature = private_key.sign(
        bytes.fromhex(hash_value),
        padding.PKCS1v15(),
        hashes.SHA256()
    )

    return signature.hex()
```

- *List. 3 – Code listing [2].*

- ***Description***

Funkcja sign_hash_with_private_key podpisuje wartość skrótu (hash) za pomocą klucza prywatnego RSA. Hash jest podawany w formie ciągu szesnastkowego (hex), a klucz prywatny jest wczytywany z pliku w formacie PEM. Wynikowy podpis jest

zwracany jako ciąg szesnastkowy.

```python
def verify_signature_with_public_key(hash_value: str, signature: str, public_key_path: str) -> bool:
    with open(public_key_path, 'rb') as key_file:
        public_key = serialization.load_pem_public_key(
            key_file.read(),
            backend=default_backend()
        )
    try:
        public_key.verify(
            bytes.fromhex(signature),
            bytes.fromhex(hash_value),
            padding.PKCS1v15(),
            hashes.SHA256()
        )
        return True
    except Exception as e:
        print(f"Verification failed: {e}")
        return False
```

- *List. 3 – Code listing [2].*

- ***Description***

Funkcja verify_signature_with_public_key weryfikuje poprawność podpisu cyfrowego przy użyciu klucza publicznego RSA. Przyjmuje hash (hex), podpis (hex) i ścieżkę do klucza publicznego w formacie PEM. Zwraca True, jeśli podpis jest poprawny, lub False w przeciwnym razie.

```python
def sign_pdf_using_private_key(input_pdf_path: str, output_pdf_path: str, private_key_path: str) -> None:
    shutil.copy(input_pdf_path, output_pdf_path)
    if(find_signature(output_pdf_path) is not None):
        print("Signature already exists.")
    else:
        create_pdf_placeholder(input_pdf_path, output_pdf_path, SIGNATURE_LENGTH_HEX)
    signature_positon = find_signature(output_pdf_path)

    hash = calculate_sha256(output_pdf_path, signature_positon, SIGNATURE_LENGTH_HEX)
    print(f"Hash value({len(hash)}): {hash}")
    signature = sign_hash_with_private_key(hash, private_key_path)
    print(f"Signature({len(signature)}): {signature}")

    replace_bytes_at_offset(output_pdf_path, signature_positon, signature.encode(), output_pdf_path)
    print(f"Signed PDF: {input_pdf_path} to: {output_pdf_path}")
```

- *List. 3 – Code listing [2].*

- ***Description***

Funkcja sign_pdf_using_private_key podpisuje cyfrowo plik PDF za pomocą klucza prywatnego RSA. Najpierw kopiuje PDF na podaną ścieżkę wyjściową, sprawdza, czy istnieje placeholder podpisu, a jeśli nie, tworzy go. Następnie oblicza hash dokumentu (z wyłączeniem placeholdera), podpisuje hash kluczem prywatnym i wstawia podpis w miejsce placeholdera.

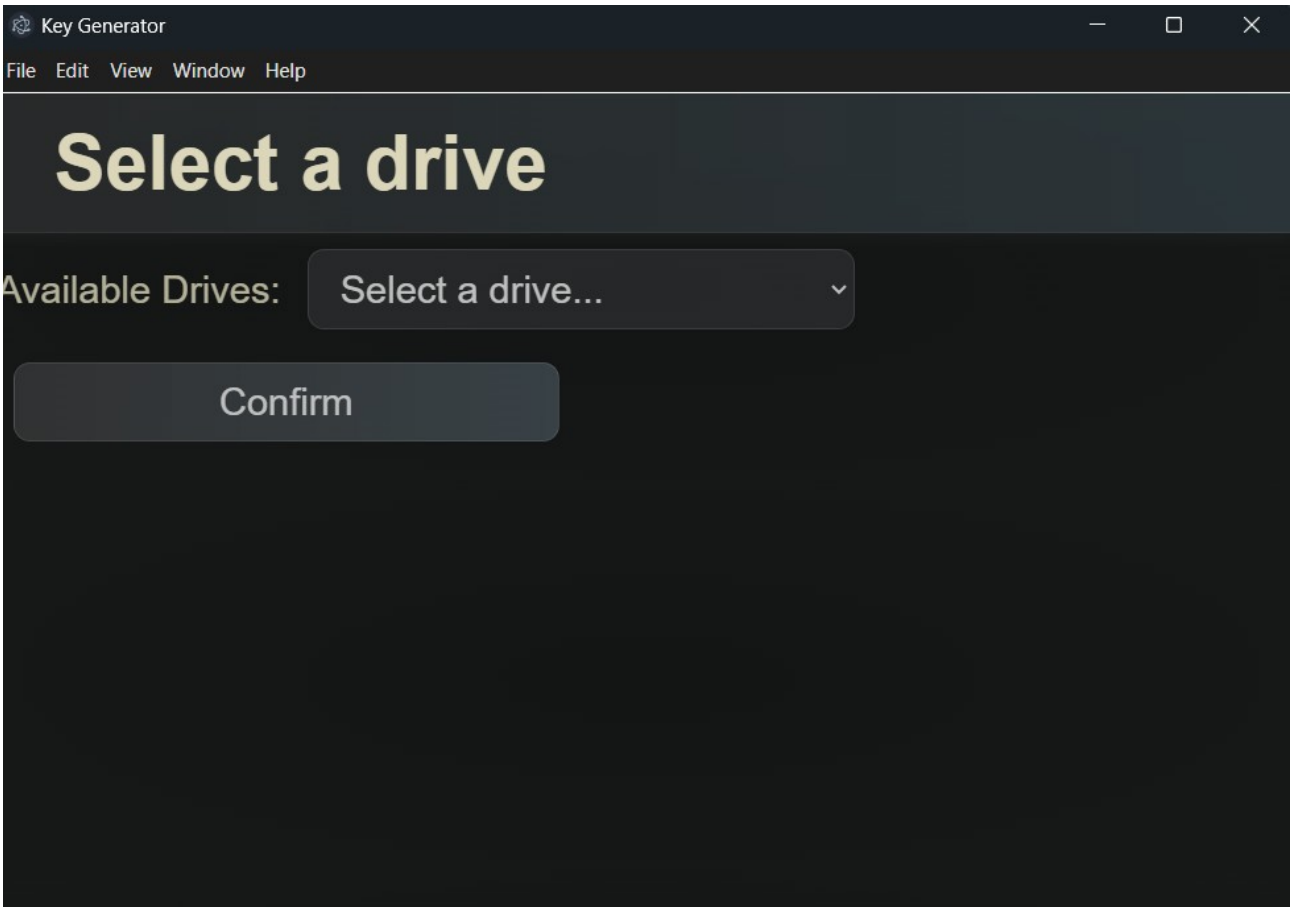
```python
def verify_signed_pdf(input_pdf_path: str, public_key_path: str) -> bool:
    signature_positon = find_signature(input_pdf_path)
    if signature_positon is None:
        print("Signature not found.")
        return False

    hash = calculate_sha256(input_pdf_path, signature_positon, SIGNATURE_LENGTH_HEX)
    signature = get_signature(input_pdf_path, signature_positon, SIGNATURE_LENGTH_HEX)
    print(f"Hash value({len(hash)}): {hash}")
    print(f"Signature({len(signature)}): {signature}")

    verified = verify_signature_with_public_key(hash, signature, public_key_path)
    print(f"Signature verified: {input_pdf_path} with: {public_key_path}")
    return verified
```

List. 3 – Code listing [2].

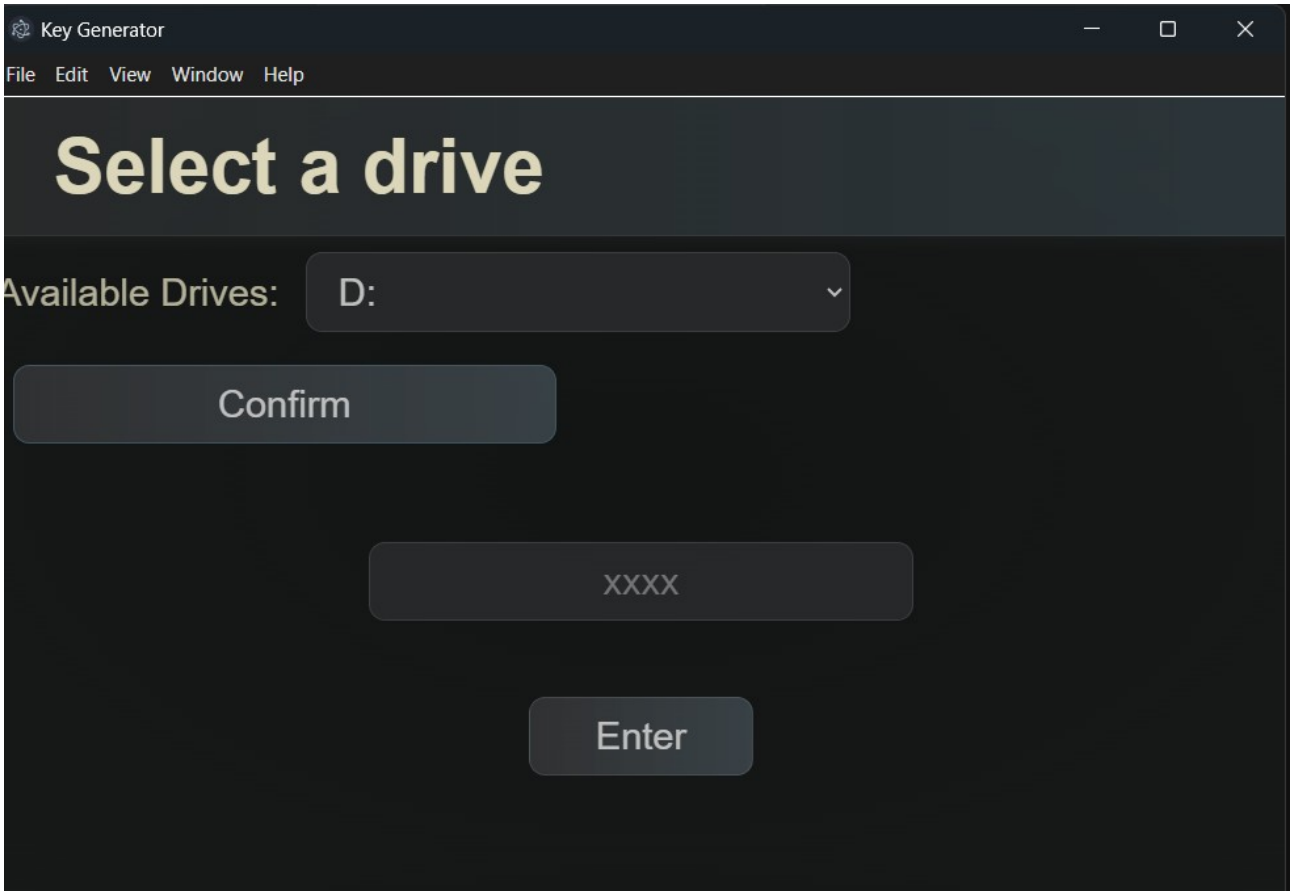
- ***Description***

  Funkcja verify_signed_pdf weryfikuje podpis cyfrowy w pliku PDF przy użyciu klucza publicznego RSA. Sprawdza obecność placeholdera podpisu w PDF, oblicza hash dokumentu (z wyłączeniem podpisu), wyciąga podpis i weryfikuje jego poprawność względem hash-a. Zwraca True, jeśli podpis jest poprawny, lub False w przeciwnym razie.

- ***Results***

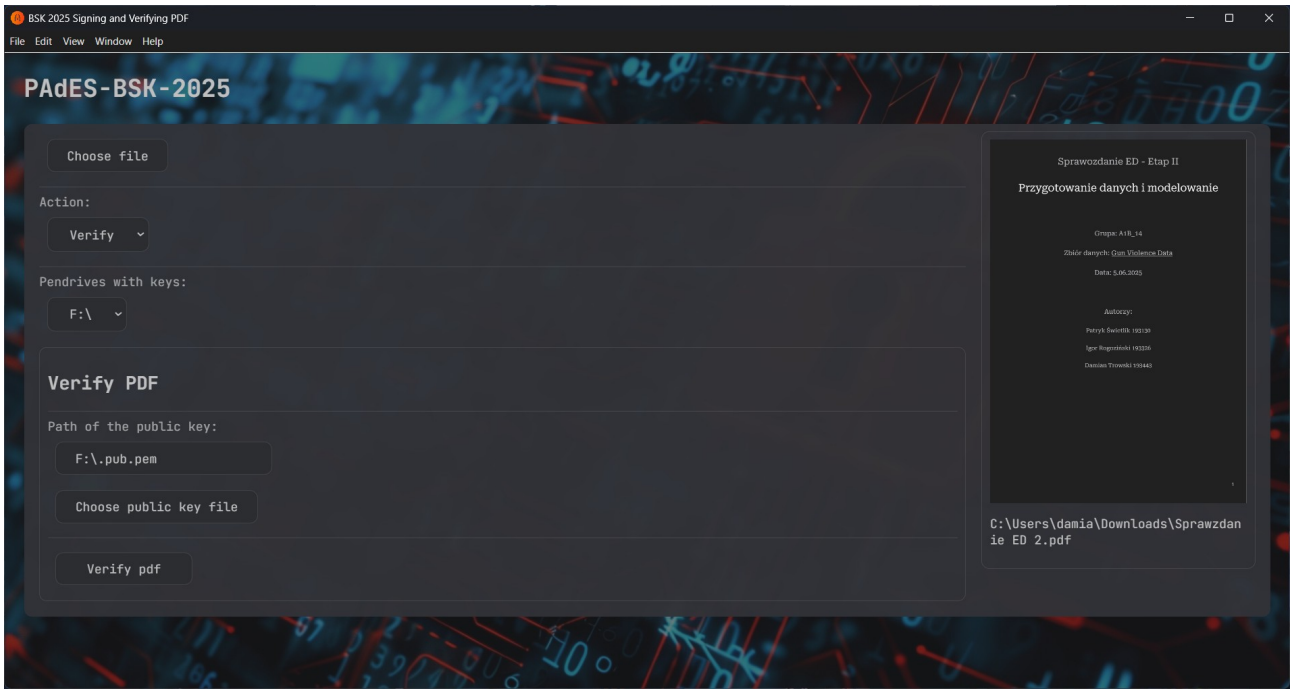


Aplikacja do generowania pary kluczy za pomocą listy wybueramy na którym dysku chcemy klucze wygenerować.



Następnie podajemy kod czterocyfrowy PIN, który służy do zaszyfrowania klucza prywatnego algorytmem AES.

Aplikacja do podpisywania plików PDF. Wybieramy plik PDF, który chcemy podpisać, a następnie wskazujemy klucz prywatny znajdujący się na wybranym nośniku. Po zatwierdzeniu aplikacja podpisuje dokument cyfrowo i zapisuje go w wybranej lokalizacji.



Aplikacja do weryfikacji podpisu cyfrowego PDF. Wybieramy plik PDF do weryfikacji, opcjonalnie wskazujemy inny klucz publiczny (lub korzystamy z domyślnego), a następnie klikamy "Verify". Aplikacja sprawdza, czy podpis w pliku PDF jest poprawny, używając wybranego klucza publicznego.

- ***Summary***

  Projekt został zrealizowany używając frameworku electron. System został podzielony na dwie aplikacje jedna generująca parę kluczy jeden do podpisywania dokumentów i drugi do weryfikacji podpisu dokumentu. Drugi komponent systemu służy do podpisywania pdf oraz do weryfikacji podpisu dokumentu.

  Repozytorium: https://github.com/d4m14n-gh/PAdES-BSK-2025

  Doxygen Documentation: https://github.com/d4m14n-gh/PAdES-BSK-2025

• Literature

[1] Article.

[2] Online Doxygen documentation, https://www.doxygen.nl/manual/lists.html, (accessed on 01.02.2025).

[3] Book.