
Bezpieczeństwo Systemów Komputerowych

Raport Projektu

Autorzy:
Anna Krasodomska, 188863
Alicja Wagner, 188971

Wersja: 1.5

Wersje

Wersja	Data	Opis zmian
1.0	18.04.2024	Utworzenie projektu i uzupełnienie opisu projektu
1.1	19.04.2024	Uzupełnienie części <i>Kod źródłowy</i> i napisanie podsumowania
1.2	22.04.2024	Uzupełnienie części <i>Graficzny interfejs użytkownika</i>
1.3	02.06.2024	Uzupełnienie podrozdziałów 2.1 oraz 2.2
1.4	04.06.2024	Uzupełnienie podrozdziałów 2.3, 2.4, 2.5 oraz 2.7
1.5	05.06.2024	Uzupełnienie podrozdziału 2.6

1. Projekt – termin kontrolny

1.1 Opis

Główną funkcjonalnością aplikacji jest możliwość wygenerowania podpisu elektronicznego zgodnego ze standardem XAdES. Ponadto możliwe jest sprawdzenie wiarygodności podpisu przez innego użytkownika, który dysponuje dokumentem xml (podpisem), dokumentem podpisywanym oraz kluczem publicznym autora. Aplikacja umożliwia także szyfrowanie i deszyfrowanie dokumentów o rozszerzeniach .pdf oraz .cpp.

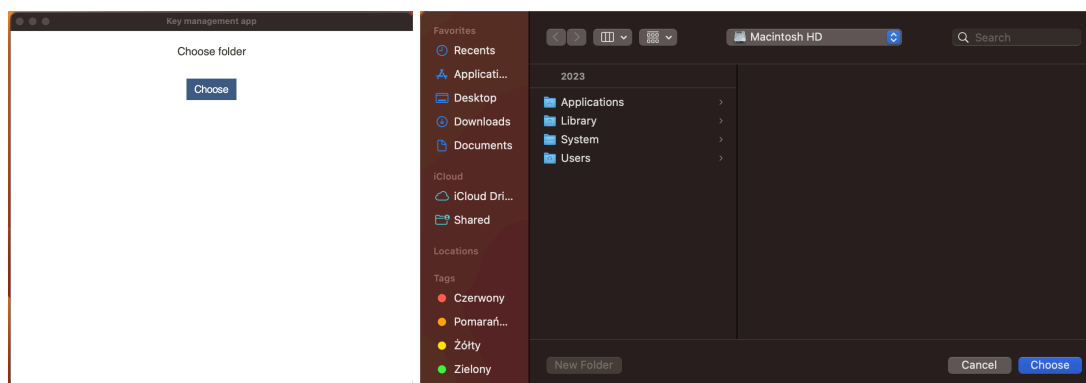
Druga, mniejsza aplikacja, służy do generowania klucza prywatnego i publicznego za pomocą algorytmu RSA. Dodatkowo, klucz prywatny jest szyfrowany za pomocą pinu podawanego przez użytkownika, a następnie oba klucze są zapisywane w katalogu. Aplikacja ta imituje działanie zaufanej trzeciej strony.

Aplikacja wykonana jest w języku Python z użyciem biblioteki Tkinter.

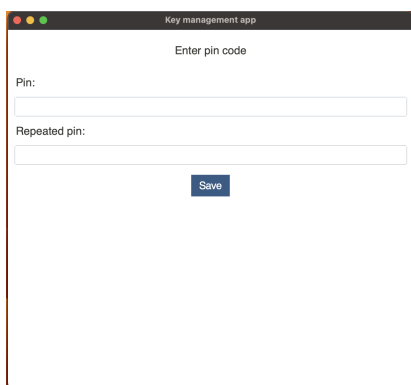
1.2 Postęp prac

1.2.1 Graficzny interfejs użytkownika

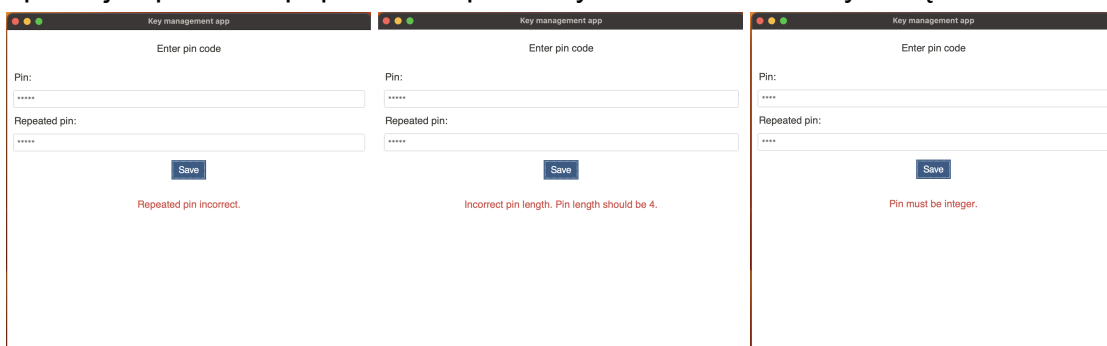
Po włączeniu aplikacji użytkownik powinien wskazać folder, w którym powinien być zapisany klucz. Po kliknięciu “Choose” otwiera się okno wyboru folderu.



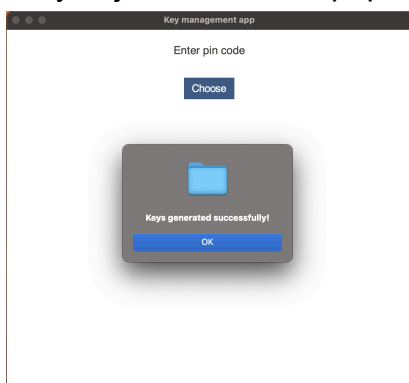
Następnie użytkownik powinien wpisać pin.



Aplikacja sprawdza poprawność pinu i wyświetla komunikaty o błędzie.



Jeśli pin jest poprawny, a klucze zostały wygenerowane poprawnie, to użytkownik otrzymuje komunikat o poprawnie wygenerowanych kluczach.



1.2.2 Kod źródłowy

Jedną z ważniejszych, zaimplementowanych już, funkcjonalności aplikacji pomocniczej jest szyfrowanie klucza prywatnego. Odbывается to przy pomocy pinu, który wcześniej musi wprowadzić użytkownik. Z pinu obliczany jest skrót, do czego wykorzystywany jest algorytm SHA-256. Tak obliczony skrót staje się kluczem dla algorytmu AES, którym szyfrowany jest klucz prywatny (w kodzie oznaczony jako zmienna *data*). Przed przeprowadzeniem szyfrowania do danych dopisywane są zerowe bajty, które mają na celu dopasowanie rozmiaru danych do wielokrotności rozmiaru bloku. Następnie tworzony jest szyfr korzystający z trybu CBC i dane są szyfrowane. Cały proces przedstawiony jest na list. 1.

```
key = hashlib.sha256(self.pin_code.encode('utf-8')).digest()
data = self.private_key
padded_data = data + b"\0" * (AES.block_size - len(data) %
AES.block_size)
cipher = AES.new(key, AES.MODE_CBC, self.IV)
encrypted_data = cipher.encrypt(padded_data)
```

List. 1 – Szyfrowanie klucza prywatnego.

W głównej aplikacji można już znaleźć implementację podpisywania dokumentu. Zaszyfrowany skrót dokumentu powstaje przy użyciu funkcji *sign()* z biblioteki *cryptography.hazmat.primitives.asymmetric*. Jest ona metodą klasy zajmującej się zarządzaniem kluczami prywatnymi. Wywołanie tej funkcji pokazuje list. 2.

```
signature = private_key.sign(
    document,
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)
```

List. 2 – Podpisywanie dokumentu.

1.3 Podsumowanie

Utworzono aplikację pomocniczą generującą klucz prywatny i publiczny oraz szyfrującą klucz prywatny za pomocą algorytmu AES. Aplikacja posiada interfejs GUI stworzony za pomocą biblioteki Tkinter.

Ponadto rozpoczęto prace nad główną aplikacją, generującą podpis cyfrowy. Zostały zaimplementowane funkcje odszyfrowujące klucz prywatny, pobierające dokument, który należy podpisać, generujące skrót z dokumentu oraz szyfrujące otrzymany skrót.

2. Projekt – termin końcowy

2.1 Wstęp

W drugim etapie projektu dokończona została funkcjonalność elektronicznego podpisywania dokumentów. Ponadto, zaimplementowane zostały nowe funkcjonalności: weryfikowanie podpisu oraz szyfrowanie i deszyfrowanie małych plików. Został także dodany graficzny interfejs użytkownika dla głównej aplikacji.

2.2 Generowanie pliku zgodnie ze standardem XAdES

XAdES (XML Advanced Electronic Signatures) to standard podpisu elektronicznego oparty na XML. Aby nie modyfikować dokumentu, który jest podpisywany, aplikacja generuje osobny plik xml zawierający informacje, takie jak nazwa osoby składającej podpis, rozmiar, rozszerzenie i datę modyfikacji pliku, znacznik czasu oraz przede wszystkim zaszyfrowany kluczem prywatnym skrót dokumentu, który stanowi jego podpis. Taki plik xml jest zapisywany w tym samym katalogu, w którym znajduje się podpisywany dokument.

Przykładowy plik xml pokazany jest na rys. 1.

```
<XAdES>
  <SignatureInfo>
    <SignerInfo>User A</SignerInfo>
    <DocumentInfo>
      <DocumentSize>18810</DocumentSize>
      <DocumentExtension>.pdf</DocumentExtension>
      <DocumentModificationDate>Sun Jun 2 15:17:34 2024</DocumentModificationDate>
      <DocumentHash>R2BrTydIqrTUz7D8Ec8oAaKvvQnJ2e0lwTH7zEv0Lr7FPmts21h9bCmQkCchgqb2CBSoaklLGnwx+SJgrRYrW
      </DocumentHash>
    </DocumentInfo>
    <Timestamp>2024-06-02T16:26:59.955015</Timestamp>
  </SignatureInfo>
</XAdES>
```

Rys. 1 – Plik xml z podpisem elektronicznym.

2.3 Weryfikowanie podpisu elektronicznego

Aby zweryfikować podpis elektroniczny, należy rozszyfrować szyfrogram skrótu dokumentu, który przechowywany jest w pliku xml, za pomocą klucza publicznego autora podpisu. Następnie należy wygenerować nowy skrót z oryginalnego dokumentu, który został podpisany. Na koniec wystarczy porównać otrzymane skróty - jeśli są one sobie równe, podpis jest prawidłowy. Dzięki takiemu sprawdzeniu mamy pewność, że dokument nie uległ modyfikacji oraz, że został on podpisany przez osobę, do której należy dany klucz prywatny i publiczny.

W celu wydobycia zaszyfrowanego skrótu dokumentu z pliku xml, używana jest funkcja pokazana na list. 3.

```
def _get_signature_from_xml(self) -> bytes | None:
    try:
        tree = ET.parse(self.signature_file_path)
        root = tree.getroot()
        document_hash_element = root.find("./DocumentHash")
        if document_hash_element is not None:
            document_hash = document_hash_element.text
            document_hash_bytes = base64.b64decode(document_hash)
            print("Encrypted hash read successfully.")
            return document_hash_bytes
    except (Exception,):
        return None
```

List. 3 – Ekstrakcja skrótu dokumentu z pliku xml.

Do porównania nowo wygenerowanego skrótu ze skrótem zaszyfrowanym za pomocą algorytmu RSA stosowana jest funkcja `verify()` z biblioteki `cryptography.hazmat.primitives.asymmetric.dsa.DSAPublicKey`, pokazana na list.4.

```
public_key.verify(
    signature,
    document,
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)
```

List. 4 – Weryfikacja podpisu.

2.4 Szyfrowanie dokumentów

Używając algorytmu RSA do szyfrowania dokumentów, trzeba wziąć pod uwagę, że możliwe będzie zastosowanie go jedynie do małych plików. Tym algorytmem można zaszyfrować tylko pliki, których rozmiar wraz z paddingiem nie przekracza długości klucza.

Funkcjonalność szyfrowania dokumentów działa podobnie do tworzenia podpisu elektronicznego. Tutaj jednak szyfrujemy kluczem publicznym, a rozszyfrowujemy prywatnym. Dzięki temu mamy pewność, że wiadomość może odczytać jedynie osoba, do której ją wysyłamy.

Na początku użytkownik musi podać ścieżkę do pliku, w którym znajduje się jego klucz publiczny. Po pobraniu klucza i odczytaniu pliku, który ma być zaszyfrowany, stosowana jest funkcja, która używa algorytmu RSA, pokaza na list. 5.

```
public_key = serialization.load_pem_public_key(
    self.public_key,
    backend=default_backend()
)
try:
    encrypted_document = public_key.encrypt(
        document,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
except (Exception,):
    raise Exception("Encryption unsuccessful. The file to be
encrypted is probably too large.")
```

List. 5 – Szyfrowanie dokumentu.

2.5 Odszyfrowywanie dokumentów

Odszyfrowywanie odbywa się z użyciem klucza prywatnego. Na początku należy wskazać miejsce, w którym zapisany jest ten klucz. Z racji tego, że jest on zaszyfrowany na potrzeby bezpieczeństwa, należy go odszyfrować, podając odpowiedni, znany użytkownikowi, pin. Po odszyfrowaniu klucza sprawdzana jest poprawność tej operacji. Jeśli wszystko przebiegło prawidłowo, można przejść do odszyfrowywania pliku.

Odszyfrowywanie odbywa się analogicznie do poprzednich przykładów, co widać na list. 6.

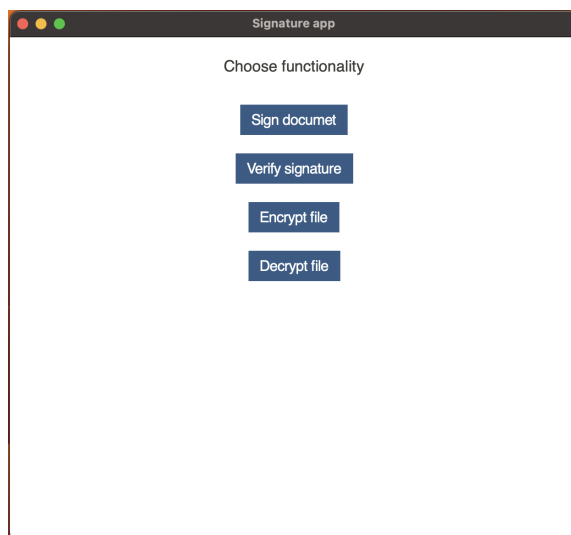
```
decrypted_document = private_key.decrypt(  
    document,  
    padding.OAEP(  
        mgf=padding.MGF1(algorithm=hashes.SHA256()),  
        algorithm=hashes.SHA256(),  
        label=None  
    )  
)
```

List. 6 – Odszyfrowywanie dokumentu.

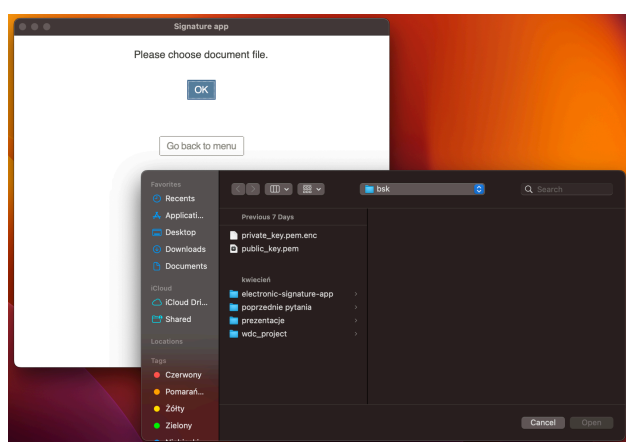
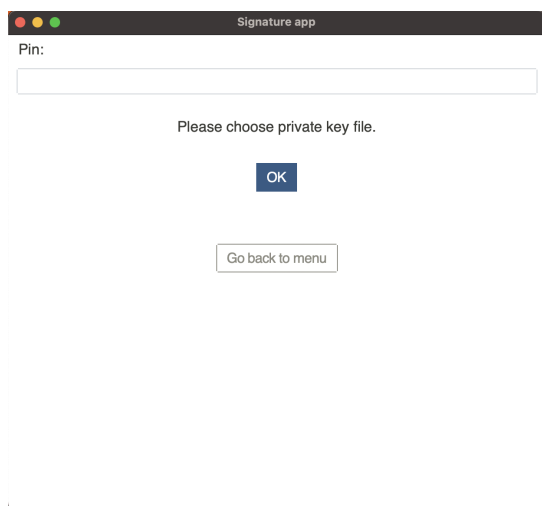
Na koniec odszyfrowany plik zapisywany jest w katalogu, w którym znajduje się także jego zaszyfrowana wersja.

2.6 Graficzny interfejs użytkownika

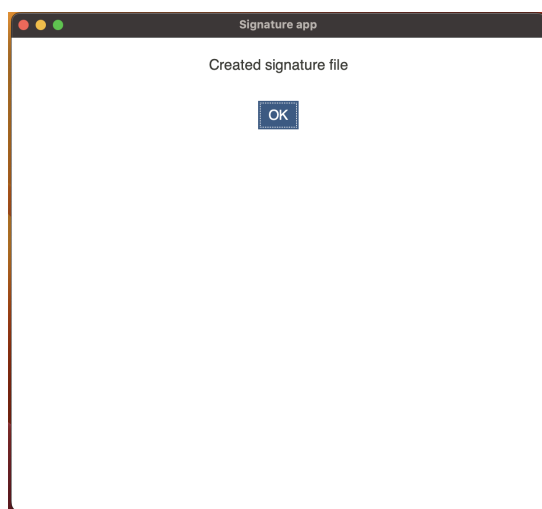
Po otwarciu programu aplikacja prosi o wybranie jednej z funkcji, którą użytkownik chce wykonać.



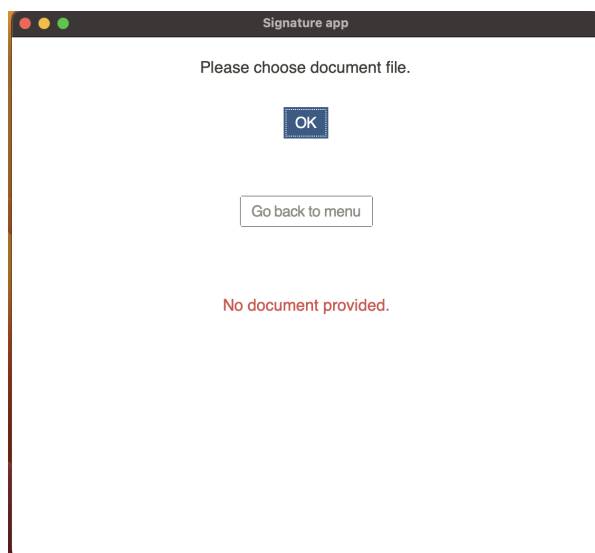
Aplikacja kolejno pyta się użytkownika o pin (jeśli jest konieczny) oraz o odpowiednie pliki informując użytkownika, który plik/pliki powinien wgrać.



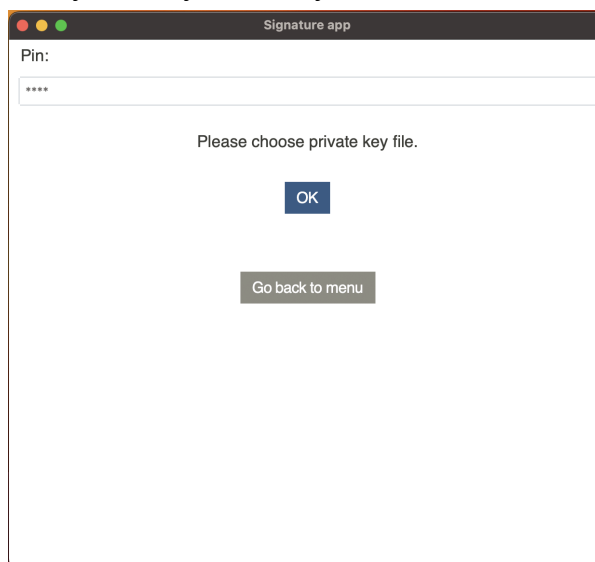
Na końcu procesu aplikacja pokazuje informacje czy proces się powiódł. Po kliknięciu "OK" aplikacja przekierowuje użytkownika do ekranu startowego.



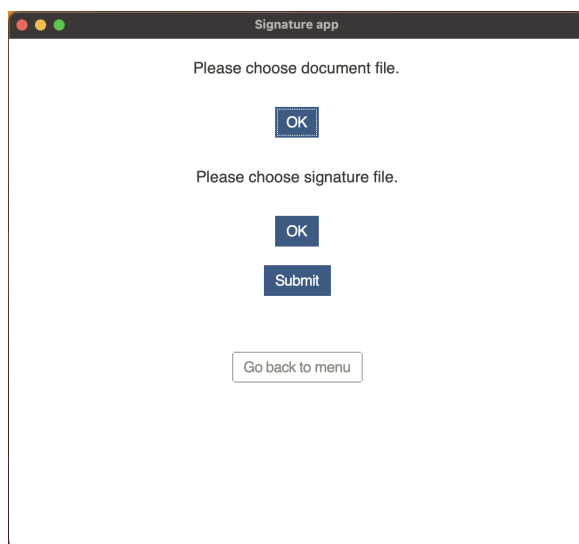
Jeśli w trakcie działania wystąpi jakiś błąd (exception) rzucony przez funkcje z backendu, aplikacja je odpowiednio obsługuje wyświetlając komunikat o błędzie na ekranie aplikacji.



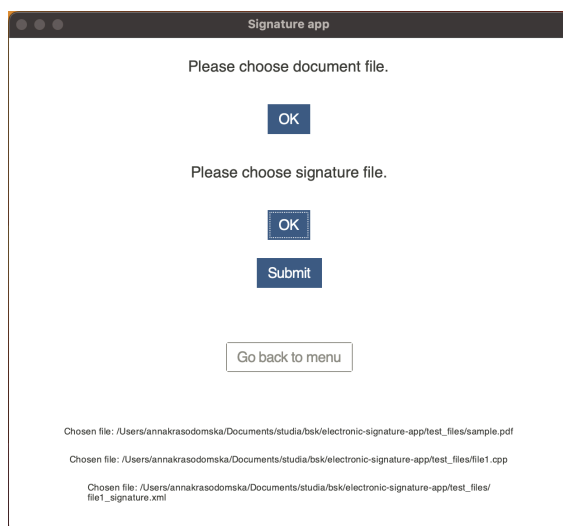
Jeśli użytkownik wybierze złą funkcjonalność, bądź rozmyśli się w trakcie, na każdym wyświetlonym ekranie ma możliwość powrotu do menu.



W przypadku wybierania kilku plików przed uruchomieniem funkcji wyświetlane jest kilka przycisków wyboru i pod spodem przycisk zatwierdzenia stanu.



Po wybraniu pliku wyświetlana jest na ekranie informacja o ścieżce do wybranego pliku.



2.7 Podsumowanie

Na potrzeby projektu utworzono 2 aplikacje: aplikację pomocniczą, generującą klucz prywatny i publiczny oraz szyfrującą klucz prywatny za pomocą algorytmu AES, oraz aplikację główną, generującą podpis elektroniczny zgodnie ze standardem XAdES, weryfikującą takie podpisy i dającą możliwość szyfrowania i odszyfrowywania małych plików.

Obie aplikacje posiadają interfejs GUI stworzony za pomocą biblioteki Tkinter.

3. Literatura

- [1] AES Encryption & Decryption In Python: Implementation, Modes & Key Management <https://onboardbase.com/blog/aes-encryption-decryption/> (dostęp 12.04.2024)
- [2] XML Advanced Electronic Signatures (XAdES) https://www.w3.org/TR/XAdES/#XML_Advanced_Electronic_Signature_Data_Structures (dostęp 13.04.2024)
- [3] RSA <https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/#signing> (dostęp 13.04.2024)
- [4] RSA (cryptosystem) [https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem)) (dostęp 28.05.2024)
- [5] Algorytm RSA w kryptografii <https://www.geeksforgeeks.org/rsa-algorithm-cryptography/> (dostęp 28.05.2024)
- [6] RSA Signatures <https://cryptobook.nakov.com/digital-signatures/rsa-signatures> (dostęp 02.06.2024)
- [7] How does RSA signature verification work? <https://crypto.stackexchange.com/questions/9896/how-does-rsa-signature-verification-work> (dostęp 02.06.2024)