

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ  
ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ  
ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»  
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ  
Кафедра інформаційної безпеки

Звіт  
з виконання завдань практичної роботи №3 на тему  
«Реалізація основних асиметричних криптосистем»  
з кредитного модуля «Методи реалізації криптографічних механізмів»

Виконали:  
студенти гр. ФБ-31мн  
Журибіда Ю.Б.  
Швець М.К.  
Шостак А.А.

Київ 2024

# Зміст

Вступ .....	3
Теоретичні відомості .....	3
Практична реалізація .....	3
Програмний код сервісу .....	7
defs.py .....	7
DES-Web-service.py .....	8
Демонстрація .....	9
Демонстрація роботи веб-сервісу через WEB UI .....	10
Демонстрація роботи веб-сервісу через cmd (curl) .....	13
Висновки .....	13

## Вступ

Мета роботи: дослідження можливостей побудови загальних та спеціальних криптографічних протоколів за допомогою асиметричних криптосистем.

Ми будемо реалізовувати завдання для підгрупи 3А — IT-систему Web-сервісу електронного цифрового підпису.

## Теоретичні відомості

Електронний цифровий підпис (ЕЦП) — це криптографічний механізм, який використовується для забезпечення автентичності та цілісності електронних документів. ЕЦП гарантує, що документ не був змінений після підписання і підтверджує особу підписанта.

Основою для створення і перевірки ЕЦП є асиметричне шифрування, яке використовує пару ключів: приватний і публічний:

- приватний ключ використовується для створення підпису. Зберігається в таємниці і доступний лише власнику;
- публічний ключ, в свою чергу, використовується для перевірки підпису. Він може бути доступний будь-кому.

Пройдемося по окремим параметрам та методам, які використаємо для реалізації завдання:

- RSA (Rivest-Shamir-Adleman) — це один з найпоширеніших алгоритмів асиметричного шифрування, який використовується для створення і перевірки ЕЦП. Він базується на математичній складності факторизації великих чисел;
- PEM — це текстовий формат, що використовується для зберігання криптографічних ключів та сертифікатів. Він містить дані у вигляді тексту, закодованого з використанням Base64, та заголовки, які ідентифікують тип даних;
- PSS — це схема доповнення, яка використовується для підпису даних в RSA. Вона забезпечує високий рівень безпеки завдяки використанню випадкових значень (сілі) та хешування;
- SHA-256 (Secure Hash Algorithm 256-bit) — це криптографічний алгоритм хешування, який створює унікальний фіксований розмір хеш для даних. Він використовується для створення хешу документа перед підписанням, що забезпечує цілісність даних.

Для спрощення реалізації веб-сервісу ми вирішили використовувати FastAPI. FastAPI — це сучасний веб-фреймворк для Python, який дозволяє швидко створювати високопродуктивні веб-додатки. Він підтримує автоматичну генерацію [якої/не якої :)] документації та перевірку типів даних та надає можливість легкого тестування сервісу.

## Практична реалізація

Структура Web-сервісу виглядає наступним чином:

```
(kali@kali)~[/DES]
$ tree
.
├── defs.py
├── DES-Web-service.py
├── env
│   ├── bin
│   │   ├── python → python3
│   │   ├── python3 → /usr/bin/python3
│   │   └── python3.12 → python3
│   ├── include
│   │   └── python3.12
│   ├── lib
│   │   ├── python3.12
│   │   └── site-packages
│   ├── lib64 → lib
│   ├── pyvenv.cfg
│   └── pycache
├── app.cpython-312.pyc
├── defs.cpython-312.pyc
└── DES-Web-service.cpython-312.pyc

10 directories, 9 files
```

Файл «defs.py» містить у собі підключення допоміжних бібліотек та визначення функцій, що будуть використовуватися в процесі роботи сервісу. Бібліотеки, що використовуються:

```
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes, serialization
```

Перша визначена функція — функція генерації ключей:

```
def generate_keys():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
    )
    public_key = private_key.public_key()

    private_pem = private_key.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.PKCS8,
        encryption_algorithm=serialization.NoEncryption()
    )
    public_pem = public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    )
    return private_pem, public_pem
```

Що робить ця функція:

- 1) генерує ключ довжиною 2048 бітів з використання публічної експоненти рівною 65537;
- 2) за допомогою методу `private_key.public_key()` отримую публічний ключ з приватного;
- 3) за допомогою методу `private_key.private_bytes` отримую байтовий рядок приватного ключа, де:
  - a. параметр `encoding=serialization.Encoding.PEM` встановлює кодування для серіалізації у вигляді PEM-у (текстовий формат для зберігання криптографічних об'єктів);
  - b. параметр `format=serialization.PrivateFormat.PKCS8` встановлює формат для серіалізації у вигляді PKCS8 (стандарт для зберігання приватних ключів);
  - c. параметр `encryption_algorithm=serialization.NoEncryption()` визначає алгоритм шифрування. В даному випадку (оскільки це все просто демонстрація роботи алгоритмів) шифрування не використовується.
- 4) за допомогою методу `public_key.public_bytes()` отримую байт стрічку публічного ключа, де:
  - a. параметр `encoding=serialization.Encoding.PEM` встановлює кодування для серіалізації у вигляді PEM-у;
  - b. параметр `format=serialization.PublicFormat.SubjectPublicKeyInfo` визначає формат для серіалізації публічного ключа;
- 5) повертає серіалізовані приватний (`private_pem`) і публічний (`public_pem`) ключі у форматі PEM.

Простіше кажучи, функція `generate_keys()` генерує пару ключів RSA (приватний та публічний), серіалізує їх у форматі PEM і повертає у вигляді байт-строк. Цей процес дозволяє легко зберігати і використовувати ключі для криптографічних операцій, таких як шифрування, розшифрування, підписання та верифікація.

Друга визначена функція — функція підписання документів:

```
def sign_document(document, private_key_pem):
    private_key = serialization.load_pem_private_key(
        private_key_pem,
        password=None,
    )
    signature = private_key.sign(
        document,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH,
        ),
        hashes.SHA256(),
    )
    return signature
```

Що робить ця функція:

- 1) приймає на вхід два параметри:
  - a. document — документ, який необхідно підписати;
  - b. private\_key\_pem — приватний ключ у форматі PEM, за допомогою якого буде створено підпис;
- 2) за допомогою методу serialization.load\_pem\_private\_key завантажує приватний ключ з байтової стрічки у форматі PEM, де:
  - a. аргумент private\_key\_pem безпосередньо містить приватний ключ у форматі PEM;
  - b. аргумент password=None вказує, що приватний ключ не захищено паролем.
- 3) за допомогою методу private\_key.sign підписує документ, де:
  - a. document — документ, який необхідно підписати;
  - b. метод padding.PSS() вказує схему доповнення PSS для підпису, де:
    - i. mgf=padding.MGF1(hashes.SHA256()) — функція генерації маски, що використовує алгоритм хешування SHA-256;
    - ii. salt\_length=padding.PSS.MAX\_LENGTH — максимальна довжина випадкової послідовності (сіль), що додається до хешу;
  - c. параметр hashes.SHA256() встановлює алгоритм хешування SHA-256, що використовується для створення хешу документа перед підписом.
- 4) повертає створений підпис у вигляді байт-строки.

Простіше кажучи, функція sign\_document() використовується для підписання документа за допомогою приватного ключа. Вона завантажує приватний ключ у форматі PEM, створює підпис для заданого документа за допомогою схеми доповнення PSS та алгоритму хешування SHA-256, а потім повертає створений підпис.

Третя визначена функція — функція перевірки підписів на документах:

```
def verify_signature(document, signature, public_key_pem):
    public_key = serialization.load_pem_public_key(public_key_pem)
    try:
        public_key.verify(
            signature,
            document,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH,
            ),
            hashes.SHA256(),
        )
        return True
    except Exception:
        return False
```

Що робить ця функція:

- 1) приймає на вхід три параметри:
  - a. document — документ, який необхідно підписати;
  - b. signature — підпис документу, який потрібно перевірити;
  - c. public\_key\_pem — публічний ключ у форматі PEM, за допомогою якого буде перевірено підпис;
- 2) метод `serialization.load_pem_public_key` завантажує публічний ключ з байт-рядку у форматі PEM;
- 3) метод `public_key.verify()` перевіряє підпис, де:
  - a. signature — підпис, який потрібно перевірити;
  - b. document — документ, для якого перевіряється підпис;
  - c. метод `padding.PSS()` вказує схему доповнення PSS для перевірки підпису, де:
    - i. `mgf=padding.MGF1(hashes.SHA256())` — функція генерації маски, що використовує алгоритм хешування SHA-256;
    - ii. `salt_length=padding.PSS.MAX_LENGTH` — максимальна довжина випадкової послідовності (сіль), що додається до хешу;
  - d. параметр `hashes.SHA256()` встановлює алгоритм хешування SHA-256, що використовується для перевірки хешу документа;
- 4) блок try-ехсепт використовується для обробки можливих винятків під час перевірки підпису.

Простіше кажучи, функція `verify_signature()` перевіряє підпис документа за допомогою публічного ключа у форматі PEM. Вона завантажує публічний ключ, перевіряє підпис, використовуючи схему доповнення PSS та алгоритм хешування SHA-256, і повертає `True`, якщо підпис дійсний, або `False`, якщо перевірка неуспішна.

Файл «DES-Web-service.py» містить у собі безпосередню програмну реалізацію веб-сервісу ЕЦП. Нас зустрічає підключення додаткових бібліотек та файлу з визначеними функціями:

```
from fastapi import FastAPI, File, UploadFile
from defs import generate_keys, sign_document, verify_signature
```

Далі ми створюємо об'єкт FastAPI, який використовується для визначення маршрутів та запуску сервісу:

```
app = FastAPI()
```

Далі ми визначаємо маршрут для POST запиту за адресою «`/generate_keys`»:

```
@app.post("/generate_keys")
def generate_keys_endpoint():
    private_key, public_key = generate_keys()
    return {"private_key": private_key.decode(), "public_key": public_key.decode()}
```

Тут `generate_keys_endpoint()` — це функція-обробник маршруту, яка:

- 1) викликає функцію `generate_keys()` для генерації приватного і публічного ключів у форматі PEM;
- 2) повертає згенеровані ключі у JSON-форматі.

Далі ми визначаємо маршрут для POST запиту за адресою «`/sign_document`»:

Тут `sign_document_endpoint()` — це функція-обробник для цього маршруту, яка:

- 1) приймає файл для підпису (`file: UploadFile`) та приватний ключ у форматі PEM (`private_key: str`);

- 2) заміщує символи `\n` на `\n` у приватному ключі для правильної інтерпретації форматування (ака нормалізація ключа на мінімалках);
- 3) зчитує вміст файлу і підписує його за допомогою функції `sign_document()`;
- 4) повертає згенерований підпис у форматі hex.

Далі ми визначаємо маршрут для POST запиту за адресою `«/verify_signature»`:

Функція-обробник цього маршруту — `verify_signature_endpoint()`, яка:

- 1) приймає файл для перевірки (`file: UploadFile`), підпис (`signature: str`) та публічний ключ у форматі PEM (`public_key: str`);
- 2) заміщує символи `\n` на `\n` у публічному ключі для правильної інтерпретації форматування (ака нормалізація ключа на мінімалках);
- 3) зчитує вміст файлу і перевіряє підпис за допомогою функції `verify_signature()`;
- 4) повертає результат перевірки (`True` або `False`) у форматі JSON.

Отже, код з цього файлу визначає веб-сервіс для генерації RSA ключів, підпису документів та перевірки підписів за допомогою FastAPI. Він забезпечує три основні маршрути:

1. `/generate_keys` для генерації ключів.
2. `/sign_document` для підпису документа.
3. `/verify_signature` для перевірки підпису документа.

Всі інші директорії та файли, відображені у виводі команди `tree`, були створені автоматично як додаткові файли середовища Python для цього проєкту.

## Програмний код сервісу

### **defs.py**

```
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes, serialization

def generate_keys():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
    )
    public_key = private_key.public_key()
    private_pem = private_key.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.PKCS8,
        encryption_algorithm=serialization.NoEncryption()
    )
    public_pem = public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    )
    return private_pem, public_pem
```

```

def sign_document(document, private_key_pem):
    private_key = serialization.load_pem_private_key(
        private_key_pem,
        password=None,
    )
    signature = private_key.sign(
        document,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH,
        ),
        hashes.SHA256(),
    )
    return signature

def verify_signature(document, signature, public_key_pem):
    public_key = serialization.load_pem_public_key(public_key_pem)
    try:
        public_key.verify(
            signature,
            document,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH,
            ),
            hashes.SHA256(),
        )
        return True
    except Exception:
        return False

```

## DES-Web-service.py

```

from fastapi import FastAPI, File, UploadFile
from defs import generate_keys, sign_document, verify_signature

app = FastAPI()

@app.post("/generate_keys")
def generate_keys_endpoint():
    private_key, public_key = generate_keys()
    return {"private_key": private_key.decode(), "public_key": public_key.decode()}

```



```

@app.post("/sign_document")

def sign_document_endpoint(file: UploadFile, private_key: str):

    private_key = private_key.replace("\\n", "\n")

    document = file.file.read()

    signature = sign_document(document, private_key.encode())

    return {"signature": signature.hex()}

@app.post("/verify_signature")

def verify_signature_endpoint(file: UploadFile, signature: str, public_key: str):

    public_key = public_key.replace("\\n", "\n")

    document = file.file.read()

    is_valid = verify_signature(document, bytes.fromhex(signature), public_key.encode())

    return {"is_valid": is_valid}

```

## Демонстрація

Для запуску проекту ми будемо використовувати сервер ASGI uvicorn. Він використовується для запуску веб-додатків, зокрема, написаних за допомогою FastAPI.

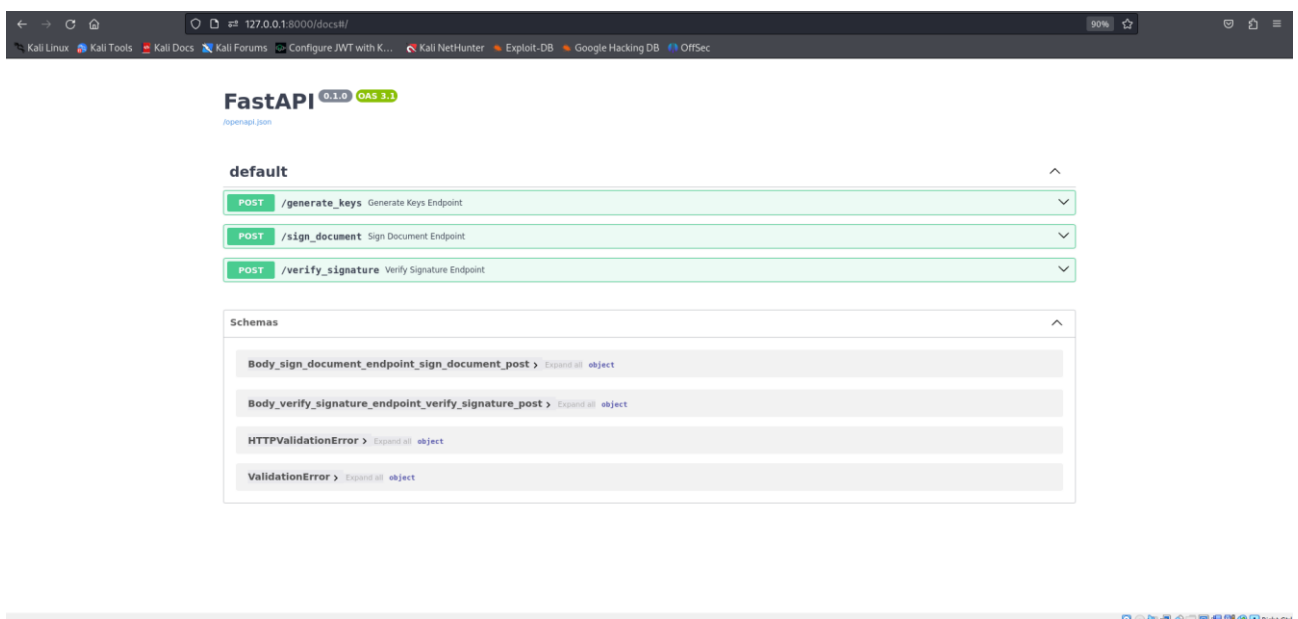
Запуск сервісу:

```

(kali@kali)-[~/DES]
$ uvicorn DES-Web-service:app --host 0.0.0.0 --port 8000 --reload
INFO: Will watch for changes in these directories: ['/home/kali/DES']
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: Started reloader process [38112] using StatReload
INFO: Started server process [38114]
INFO: Waiting for application startup.
INFO: Application startup complete.

```

Головна сторінка веб-сервісу виглядає наступним чином:



Тут ми маємо автоматично створену документацію у вигляді схем:

**Schemas**

**Body\_sign\_document\_endpoint\_sign\_document\_post** Collapse all **object**  
file\* string **binary**

**Body\_verify\_signature\_endpoint\_verify\_signature\_post** Collapse all **object**  
file\* string **binary**

**HTTPValidationError** Collapse all **object**  
detail Collapse all array<object>  
Items Collapse all object  
loc\* Collapse all array<(string | integer)>  
Items Collapse all (string | integer)  
Any of Collapse all (string | integer)  
#0 string  
#1 integer  
msg\* string  
type\* string

**ValidationError** Collapse all **object**  
loc\* Collapse all array<(string | integer)>  
Items Collapse all (string | integer)  
Any of Collapse all (string | integer)  
#0 string  
#1 integer  
msg\* string  
type\* string

А також самі реалізовані функції для POST-запитів:

**default**

<b>POST</b>	<b>/generate_keys</b>	Generate Keys Endpoint	⌵
<b>POST</b>	<b>/sign_document</b>	Sign Document Endpoint	⌵
<b>POST</b>	<b>/verify_signature</b>	Verify Signature Endpoint	⌵

## Демонстрація роботи веб-сервісу через WEB UI

Через інтерактивну панель ініціалізуємо виконання створення ключів:

**POST /generate\_keys** Generate Keys Endpoint

**Parameters** Cancel  
No parameters

Execute

Clear

**Responses**

**Curl**  
curl -X 'POST' \  
'http://127.0.0.1:8000/generate\_keys' \  
-H 'accept: application/json' \  
-d ''

**Request URL**  
http://127.0.0.1:8000/generate\_keys

**Server response**

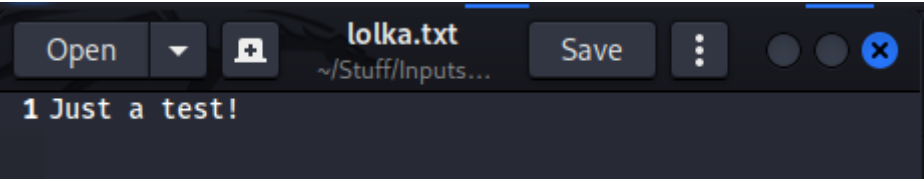
**Code** **Details**

Отримуємо наступну відповідь:

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{   "private key": "-----BEGIN PRIVATE KEY-----\nTIEvATBADA\nBgkqhkiG9w0BAQ\nEFAASCBK\nYwggSIAgEAAo\nTBAQ\nPFH56jg9ZVj2r\n7k1LZ\nHJkjdT\nhYoPthSeQ\nIXI8Gt4Z\nnW\nMlgPrdqHk\nCNfAz1Yr\nAveQh\nAF1SeB\nv\nnC4\nl0oK\nJF5fyi7\nWp21j\nWkzi\naxz8\nGoth\nCjzcf\nP\nMekh\nsujR\nXqf\nq4dd\n7y8\nR0e\nDK\n2\nBktp\n0X\nC7E\nTyz\nWhe\nym0\np4R\ntVE\nv0\nM1r\n08or\nH0oa\n2p\nzf\nNAPE\njvnT\nE7T\nfZ\nm\nPF51\nv\nn\nPCy\nak\nNM\n0\nKKX\n2\nHhg\nqtU\nHS\nfT\nCA\nS0\nCZ\n1\nHd\nWt\nu\nh\nMoF\ny\nW\nK\nWgs\n1\nB\n8\nv\nr\n4\nFA\nj\n0\nW\nY\n1\n56\n5\nLi\nC\nW\nm\nX\n2\n5\nC\n6\nM\nI\ny\nM\nF\ns\nG\nS\nI\n0\net\nL\nM\nG\nJ\na\ni\n6\n4\n0\nux\nzy\nY\nM\nU\nr\nL\nG\nC\n1\nG\nX\n1\nS\nE\nA\n0\nk\nq\nh\nB\nn\nE\nH\nF\nB\na\nP\nb\nX\nA\nq\nM\nB\nA\nE\nC\nG\nE\nA\n0\nnv\ny\n1\nE\nT\nS\nL\nr\nS\nS\nV\nR\nj\nG\nZ\nF\nD\nS\nG\nJ\nK\n6\nE\na\nz\n3\n2\n+\nG\nE\nW\nz\n0\nk\n5\n0\nb\nD\nn\nC\n6\n8\nR\nD\ny\ns\nR\nQ\nv\nD\nL\nn\nU\ni\nU\nu\nu\nQ\nX\nA\nT\nW\nb\nL\nE\n2\nH\nG\nh\nC\n1\nG\ny\nB\nm\n8\nn\nU\na\nV\nu\nX\nQ\nJ\nK\nw\na\nC\nC\nU\n3\nI\nF\nT\nE\nF\nv\nE\n\nV\nK\n0\nJ\nC\nV\nD\n0\nF\nG\nY\n2\nS\n0\nF\nl\nE\nw\nD\nH\nh\n1\nz\n6\nB\na\ny\nr\nG\nL\nM\nG\nu\nK\n5\n2\ns\n0\nE\nG\nT\n0\nW\nM\ns\nG\nL\nQ\nJ\nH\nF\nM\nD\nT\nX\nT\nK\nX\nm\nG\nR\n\nz\nj\nn\nq\nD\ny\nW\nK\n2\nD\nS\n+\nT\nD\n0\nP\n+\n9\nJ\n4\nK\n4\nB\n5\nD\n/\nD\nL\nR\nN\nv\n4\nG\nF\nN\nT\nZ\nh\nK\n1\nK\na\n5\n4\nt\ns\nK\n1\nr\n1\nH\n4\nG\nK\nC\nB\nr\n8\n\no\n/\nT\nB\nY\nD\nL\ns\nJ\n8\nG\nE\nK\n2\n5\nG\nC\nq\nX\nL\nq\nz\nD\n4\nF\nC\ny\nH\nY\nK\n3\nZ\nQ\nY\n8\nD\nE\n+\nE\nP\nJ\nF\nJ\nG\n7\nh\n+\nK\nK\nL\nf\n5\nG\nM\nG\nM\nZ\nD\n8\nL\nZ\n\nn\na\ny\nS\n2\nT\nw\nL\nu\nu\nr\nI\nX\n0\nL\nS\nj\n0\nP\nJ\nE\nD\n0\nm\n5\nh\nS\nT\nL\nN\nY\nV\nK\nf\nL\nG\nf\n6\nu\n4\nQ\nK\nB\nQ\nD\nI\n0\nE\n1\nW\nM\nu\nH\nX\nh\nq\nz\nC\n5\nQ\nL\nF\n\nn\nk\nL\nI\n7\nf\n/\nG\nL\nF\nH\nX\nP\nY\nF\nr\nD\nD\nI\nq\nh\nq\nF\nM\n0\nz\ny\n9\nE\nT\na\nJ\n3\n0\nS\nG\nP\nX\n8\nK\nw\nD\nw\n2\n8\nM\nq\nK\nC\nD\n9\n2\nD\ny\nG\nL\nS\n+\nL\nT\nS\n8\n\nf\nX\nB\np\ni\nq\nD\n/\nL\nC\n3\ns\n2\nx\na\n8\nI\n6\nL\n3\nG\nY\nm\nL\nz\nv\nI\nF\nZ\n5\nE\nK\nu\ns\nx\nh\nC\nF\nD\n6\ns\ny\nK\n3\nV\nD\n7\nr\nM\nu\nV\nr\n7\n0\nK\nF\n7\nv\n9\n\na\nn\na\n2\n0\n8\nz\nT\np\na\nX\n5\n0\nS\nF\nZ\nH\nz\nD\nf\np\n4\nr\nK\nB\nQ\nD\nE\nF\nV\nV\nx\nT\nf\n0\n9\nD\n8\n0\nN\nT\n4\n0\nI\n5\n5\nK\nT\n0\n0\nT\nA\n4\n6\nC\nr\nz\nQ\nZ\n\nn\nV\nC\nv\nK\nM\nB\nD\nX\nn\nS\nB\n0\nQ\nG\n6\nR\nL\nA\nT\nN\nv\nD\nE\nS\nG\nR\n/\nL\nK\n1\n3\nG\n1\n7\nf\nG\n7\n8\n4\n0\nG\na\nE\nA\nD\nj\nn\n+\nZ\nM\nC\n7\nA\nl\n0\nm\nh\n\nu\nm\nZ\nf\nQ\nE\nu\nC\nA\nZ\n0\n9\nF\n5\nj\n0\nw\nD\nF\nW\nY\nC\n7\nJ\nU\nJ\nM\n0\nB\n9\n+\nE\nS\n3\nI\nX\n0\nz\nJ\nn\nX\nG\nW\nY\nL\nC\n/\n6\n1\n+\nJ\nG\nT\nE\nG\n0\nC\n0\n\nn\n0\n0\nz\nq\nq\n-\nD\nE\n-\nK\nB\nQ\nD\n/\nX\nL\nL\nN\nL\nF\nE\nM\nI\nS\nM\nS\n3\nw\n-\nQ\nK\nM\nC\nS\nJ\n2\nC\nB\nM\nv\nG\nP\nH\nT\nI\nZ\nh\n8\nr\n+\nS\n5\nY\nF\n0\n\nn\n2\nT\nU\nE\n5\nh\nT\nI\n0\nX\nv\n8\nT\n0\nI\nI\nC\nF\nK\n3\nZ\n1\nX\nZ\nF\nA\nU\nS\nU\n0\nT\n2\nE\nB\nr\nS\nP\nS\nM\nZ\nQ\n9\nK\nZ\nY\nu\n+\nS\nk\nI\nq\nh\nu\nK\n3\nN\nI\nI\nC\nE\n\nn\n8\n7\n0\nb\nk\nw\nZ\nF\nE\nr\nV\n9\nX\nV\nR\nD\nX\n/\nV\nQ\nR\nj\nX\nB\nh\ni\nL\ng\nz\nr\nI\nT\nX\nD\nr\nh\nY\nO\nV\nj\nY\nI\n0\nF\nG\nW\nA\nZ\n0\nC\nL\np\nA\n0\nG\nA\nM\nu\ns\n\nn\nS\nW\nu\nP\nK\nF\n1\nB\nM\n7\nF\n5\nn\n0\nW\nj\nP\n3\n/\nG\ni\nb\nE\nC\nT\nV\ni\n4\nJ\nR\nP\nf\nj\n3\nX\nZ\n1\nm\nP\nM\n4\nY\nX\n2\nH\nF\nG\nD\nf\nG\n4\nh\nS\nC\nI\n\nn\nZ\nU\nV\nK\nJ\nQ\nC\nS\nE\nq\n+\nF\ny\n+\nK\nu\nX\nb\n4\nE\nC\ni\n5\n7\nE\ni\nH\nr\n6\n+\nB\n3\ni\nY\n0\nP\nH\ni\nP\nB\nQ\n7\nX\n6\n2\n0\nK\nC\nz\nY\nL\n8\nC\nP\nY\nG\nB\nW\nR\nX\nR\nC\n\nn\nM\n6\n0\nJ\nM\nL\n/\nH\nS\nF\nS\nr\nn\nv\nC\nD\nS\n3\n8\nK\n7\nM\n/\nE\nh\nj\n6\nP\n1\n/\nB\nb\n9\nA\nP\nE\nK\nC\nY\nB\nL\nC\nI\n0\nY\nX\nA\nL\nw\n/\nP\nR\n6\n8\nL\nN\nL\nS\nS\nB\n\nn\nk\nv\nr\nI\nC\nE\n+\nN\nQ\nW\n9\nH\ni\n8\nr\nj\n9\n0\nD\nE\nG\nH\nS\n8\n3\n3\nM\nL\n7\nU\nU\n1\nV\nQ\nU\nX\nI\n/\n1\nC\nL\n/\nQ\nv\nh\nN\nU\nh\n0\nM\nD\n0\nw\nh\nh\nv\nC\nD\nV\nX\n\nn\nV\no\nJ\n0\nn\n/\nI\nn\nI\n2\nT\nQ\nL\n3\nV\nR\nZ\nI\nS\nI\nK\n4\nG\nY\n/\nM\nH\nD\n8\nD\nn\nQ\nW\na\nR\nC\nO\nF\nE\nZ\nD\n+\nF\nP\nP\n8\nP\nI\n0\nW\nP\nG\nR\n1\nY\nu\nY\n0\nP\nX\nN\nS\n\nn\n6\n0\n9\nH\nT\n0\n/\nI\nq\nI\n8\n/\nN\nT\nK\n9\nS\nD\nn\nj\nA\n=\n\nn\n-----\nEND PRIVATE KEY\n-----\n\n",   "public key": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQAMIAIBCGCAQEAxR+ueo7PWVY9q+5CC2TC\n\nn\nS\n0\n3\nH\nM\nU\n2\nK\nD\n7\nY\nU\nn\nK\nD\nI\nF\ny\nP\nB\nI\nH\nM\nb\nZ\nj\nY\nD\n6\n3\na\nh\n5\nA\nP\nD\nX\nM\n5\nW\nK\nw\nL\n3\nk\nI\nQ\nB\nZ\nU\nr\nL\nW\nu\nH\nJ\na\nB\ni\n\nI\nT\n+\nR\nC\n0\nu\nZ\nB\nD\nY\nX\nS\n4\nI\nm\nS\nC\n9\nM\nR\nQ\nL\nT\nQ\n0\n8\n3\nB\nZ\nZ\nX\nP\nI\nD\nb\nL\n0\nV\n6\n4\n0\nH\nX\nE\n8\nV\nE\nD\nH\ng\nY\nT\n0\nV\n7\na\nD\nF\n\nn\n3\n0\nX\nE\n8\nm\nC\nI\nR\nM\nP\nj\n0\nM\nO\nE\nB\nV\nR\nL\n9\n8\n9\na\n0\nP\na\nK\nX\n6\nK\nG\nm\na\nC\n3\nx\nu\nD\nX\nI\n7\n5\n0\n+\nU\nE\n7\nX\nZ\nZ\nR\nE\nu\nS\nL\nz\nw\ns\nm\nP\nC\nj\n\nn\nF\nj\nn\nS\ni\nK\nD\nj\nB\n4\nK\nr\nV\nB\n+\nX\n0\nX\nG\nE\nj\nh\nX\nM\nS\nI\nX\nT\nI\nh\nv\n4\nV\n0\nR\nh\nC\ni\nL\nv\nI\nn\nI\nT\nG\nC\nf\nL\n6\n+\nB\nQ\nC\nX\nT\nf\nS\nU\nu\nu\n4\nG\n\nn\nl\nP\n1\n8\n9\nu\nQ\nU\nj\nC\nM\nl\nP\nh\nB\n8\nI\nK\nH\nr\nS\nZ\nB\nI\nW\n0\nu\nM\nD\nr\nS\nC\n8\nq\nD\nF\nK\nY\n4\nB\nT\nR\nV\n9\nU\nh\nA\nH\nK\nJ\nI\nI\nQ\nR\nB\nX\nQ\nW\nj\n2\n\nn\nl\nw\nI\nD\nA\nQ\nA\nB\n\nn\n-----\nEND PUBLIC KEY\n-----\n\n"</pre></div><div>Response headers</div><div><pre>content-length: 2226 content-type: application/json date: Wed, 20 Nov 2024 11:48:25 GMT server: uvicorn</pre></div></div>

Створили файл, який хочемо підписати щоб в майбутньому підтверджувати його автентичність:



Через інтерактивну панель ініціалізуємо виконання підпису обраного файлу з нашим приватним ключем:

POST /sign\_document Sign Document Endpoint

Parameters

Cancel

Reset

Name	Description
private_key * required	
string (query)	:9SdnjIA=\\n-----END PRIVATE KEY-----\\n

Request body required

multipart/form-data

file \* required

string(\$binary)

Browse... lolka.txt

Execute

Отримуємо наступну відповідь з нашим цифровим підписом:

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{   "signature": "bf89d22ac6618f510c6be27c152ca0fa51d3fd177d4c48b1041f7bfce6fa5499c3d0f3242946eb0640621d4788910ed5e8cbc8697130e34864355d1ec45af649d892cc3e31a1b95b5798481062b773b\n\nb\nD\n2\n1\n6\n5\n3\n5\nf\n1\n9\n7\nf\n5\nC\n0\nS\n6\nS\nd\nD\n2\n0\n0\nC\n+\nS\n3\n0\n2\nr\nE\n9\n2\nC\n4\n0\nf\nb\n3\n7\n1\nB\n9\n1\nC\n2\nf\n6\nH\nF\n5\n7\nb\nf\n6\nE\nD\n4\n2\n0\nF\n4\nE\n0\nC\n1\n4\n4\n4\n7\n0\n6\nC\n6\n1\nb\nD\n6\nE\nf\n8\n4\nb\n4\n2\n1\n3\n4\n2\n2\n0\n0\n2\n+\n2\n4\n3\nf\n9\n3\n0\nB\n9\n0\nb\nD\nE\nD\n0\nf\n0\na\n8\n7\nC\n9\n0\nE\n6\n9\n3\n7\n3\nD\nE\n9\nC\n4\nf\n0\nD\n5\nC\nb\n3\n8\nf\n0\nb\n+\nS\nC\nH\nA\nR\n+\nI\nG\nD\n5\na\n1\n4\n3\n5\nE\n2\nf\nb\nD\n1\nE\n8\n5\n9\nf\n4\n0\n7\n2\na\n8\n3\nf\n3\n0\n0\n0\na\nE\n6\n0\n5\na\n5\nb\n6\n9\nD\nf\n0\n4\n4\n2\n3\nE\n8\nD\n4\n8\n5\nE\n2\na\n0\nF\n5\n6\n2\n2\nD\n3\nE\n7\n6\n5\n3\nD\nE\nf\n0\nC\n2\n0\n2\nb\n0\n0\n3\n9\nC\n7\n4\nC\nD\na\nF\n8\n7\nC\n4\nD\nC\n5\n7\nC\n7\nE\n5\nb\n2\n8\nf\n7\nb\n1\nf\n0\n7\nD\nE\n0\n5\n9\n4\nb\nf\n0\nE\n8\n3\n5\n1\n2\nC\nD\n2\nf\n1\n6\n9\nC\nB\n8\n4\n4\n\n\n"</pre></div><div>Response headers</div><div><pre>content-length: 528 content-type: application/json date: Wed, 20 Nov 2024 11:52:20 GMT server: uvicorn</pre></div></div>

Через інтерактивну панель ініціалізуємо виконання перевірку підпису обраного файлу з публічним ключем:

POST

/verify\_signature

Verify Signature Endpoint

Parameters

Cancel

Reset

Name	Description
<b>signature</b> * required	
string (query)	35122cdd2f169cb8d4aad8e442c2fa1fd
<b>public_key</b> * required	
string (query)	\n1wIDAQAB\n-----END PUBLIC KEY-----\n

Request body

required

multipart/form-data

**file** \* required

string(\$binary)

Browse...

lolka.txt

Execute

Отримали наступну відповідь, що свідчить про валідність підпису:

Server response

Code

Details

200

Response body

```
{
  "is_valid": true
}
```

Download

Response headers

```
content-length: 17
content-type: application/json
date: Wed, 20 Nov 2024 11:57:15 GMT
server: uvicorn
```

Спробуємо змінити 1 символ в публічному ключі. Отримали наступну відповідь:

Server response

Code

Details

200

Response body

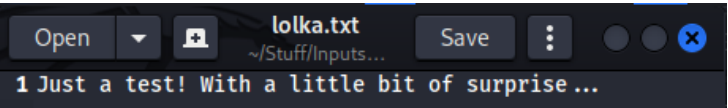
```
{
  "is_valid": false
}
```

Download

Response headers

```
content-length: 18
content-type: application/json
date: Wed, 20 Nov 2024 11:58:35 GMT
server: uvicorn
```

Внесемо зміни у файли:



Спробуємо з валідними підписом та публічним ключем провести перевірку над цим файлом. Отримали наступну відповідь, що свідчить про невідповідність підпису до цього файлу:

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/verify_signature?signature=bf89d22ac6618f510c6be27c152ca0fa51d3fd177d4c48b1041f7bfc6e6a5499c3d0f3242946b0040621d47889419ed5e8cbc8697130e34864355d1ec45af649d892cc3e31a1b95b5798481062b773bb2126356f197f5cc838e5dd2d0acc5302fcee2c49fb371b91c2f6df0557b7f6e2d4209f4ebc144df686c61bb0cf94bb2f2a2f2bb02c24b3f9380890b0ed0f0a97c98e693737d4c34f053eb38f63de5a808e1960cded5aa143f85c2f1ad858f48072a0373808a8e0955b89df8e443ae68485e2a0f56220d5e7653bdefa8cc202bb039cb74cdafa87c4dc577e95b28f87bb1f07de85d94bf08e835122cdd2f169cb8d4aad8e442c2fa1fd&public_key=-----BEGIN%20PUBLIC%20KEY-----%5CnMIIBjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxN2Bueo7PMVY9qg\2B5CC27C5Cn5o3HMU22KD7YUnkD1FyPBihMbZyJJYD63ah5ApDXw5WKwL3k1Q8ZUrNLwuH3aBk55Cn17ZBRcouzD8dYxys4Imsc9MRqLTQ83B2zzXpIdLo0V6n4OHxe8EdHgytoV7adF35Cn30xE8ncIRMpjoMDEbVRL9B9a0PaKx6Gmac3xwDx1750e2BUe7X2ZzReusLzwsmpCj\5CnFjn51kdjB4KvB82BX0xgEjhmSIXTLbv4VqBbcjLyLoLgcfL6%2B8QCXfstUuu54g5Cnlp189uQqjCHtphbtk1KfrSzBiNoum0Brsc8qubFKy48tRv19uAhMj11QRBxQj245Cn1wIDAQAB5Cn-----END%20PUBLIC%20KEY-----%5Cn'
```

Request URL

```
http://127.0.0.1:8000/verify_signature?signature=bf89d22ac6618f510c6be27c152ca0fa51d3fd177d4c48b1041f7bfc6e6a5499c3d0f3242946b0040621d47889419ed5e8cbc8697130e34864355d1ec45af649d892cc3e31a1b95b5798481062b773bb2126356f197f5cc838e5dd2d0acc5302fcee2c49fb371b91c2f6df0557b7f6e2d4209f4ebc144df686c61bb0cf94bb2f2a2f2bb02c24b3f9380890b0ed0f0a97c98e693737d4c34f053eb38f63de5a808e1960cded5aa143f85c2f1ad858f48072a0373808a8e0955b89df8e443ae68485e2a0f56220d5e7653bdefa8cc202bb039cb74cdafa87c4dc577e95b28f87bb1f07de85d94bf08e835122cdd2f169cb8d4aad8e442c2fa1fd&public_key=-----BEGIN%20PUBLIC%20KEY-----%5CnMIIBjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxN2Bueo7PMVY9qg\2B5CC27C5Cn5o3HMU22KD7YUnkD1FyPBihMbZyJJYD63ah5ApDXw5WKwL3k1Q8ZUrNLwuH3aBk55Cn17ZBRcouzD8dYxys4Imsc9MRqLTQ83B2zzXpIdLo0V6n4OHxe8EdHgytoV7adF35Cn30xE8ncIRMpjoMDEbVRL9B9a0PaKx6Gmac3xwDx1750e2BUe7X2ZzReusLzwsmpCj\5CnFjn51kdjB4KvB82BX0xgEjhmSIXTLbv4VqBbcjLyLoLgcfL6%2B8QCXfstUuu54g5Cnlp189uQqjCHtphbtk1KfrSzBiNoum0Brsc8qubFKy48tRv19uAhMj11QRBxQj245Cn1wIDAQAB5Cn-----END%20PUBLIC%20KEY-----%5Cn'
```

Server response

Code

Details

200

Response body

```
{
  "is_valid": false
}
```

Download

Response headers

```
content-length: 18
content-type: application/json
date: Wed, 20 Nov 2024 12:00:37 GMT
server: uvicorn
```

Генерація ключів:

### Підписання документу:

### Перевірка підпису:

### Перевірка підпису на іншому файлі з тим самим підписом:

## Висновки

Реалізація веб-сервісу для роботи з електронним цифровим підписом вимагає розуміння криптографічних принципів, таких як асиметричне шифрування та алгоритми хешування. Важливими теоретичними аспектами є використання приватного та публічного ключів, формату PEM, схеми доповнення PSS та алгоритму хешування SHA-256.

У програмній реалізації ми розглянули ключові функції для генерації RSA ключів, підписання та перевірки підписів, а також створення веб-сервісу за допомогою FastAPI. Використання FastAPI забезпечує швидку та ефективну реалізацію сервісу для генерації ключів, підписання документів та перевірки підписів. Завдяки Unicorn, ми можемо легко запустити сервіс і зробити його доступним для користувачів.