МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Лабораторна робота №3

З дисципліни «МЕТОДИ РЕАЛІЗАЦІЇ КРИПТОГРАФІЧНИХ МЕХАНІЗМІВ»

Роботу виконали

студенти групи ФІ-21мн, ФТІ

Татенко Вадим

Хмелевський Святослав

Кірсенко Єгор

2023

Мета роботи:

Розробити реалізацію асиметричної криптосистеми.

Варіант 3А - Реалізація Web-сервісу електронного цифрового підпису.

Хід роботи:

Посилання на джерело інформації стосовно реалізації підпису та верифікації повідомлень/документів.

Для початку необхідно згенерувати private_key та public_key, які будуть використовуватись для підписання та верифікації документа відповідно. Для цього використаємо OpenSSL в PowerShell:

openssl genpkey -algorithm RSA -out private_key.pem
openssl rsa -pubout -in private_key.pem -out public_key.pem

```
PS E:\универ\6 - курс\Крипта\lab2> openssl genpkey -algorithm RSA -out private_key.pem
>> openssl rsa -pubout -in private_key.pem -out public_key.pem
```

Тепер створимо мінімальний сервіс для підписання документу та верифікації підпису.

Код програми:

```
from flask import Flask, request, jsonify
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives.serialization import
load_pem_private_key, load_pem_public_key
from cryptography.hazmat.backends import default_backend
from cryptography.exceptions import InvalidSignature

app = Flask(__name__)
```

```
# Function to check allowed file
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower()
in {'pem', 'txt'}


# Route for sign file
@app.route('/sign', methods=['POST'])
def sign():
    # Check is all necessary files sent(private_key_file and
data_file)
    if 'private_key' not in request.files or 'data_file' not in
request.files:
        return jsonify({'error': 'Missing key or data'}), 400

    # Extracting data from request
    private_key_file = request.files['private_key']
    data_file = request.files['data_file']

    # Check is files set. Send 400 error(Request failed) if they
not
    if private_key_file.filename == '' or data_file.filename == '':
        return jsonify({'error': 'No selected file'}), 400

    # Check is files have allowed format
    if allowed_file(private_key_file.filename) and
allowed_file(data_file.filename):
        # Get instance of private key from *.pem file
        private_key = load_pem_private_key(
            private_key_file.read(), password=None,
backend=default_backend()
        )
        data_to_sign = data_file.read()

        # Signing data and getting signature
        try:
            signature = private_key.sign(
                data_to_sign,
                padding.PSS(
                    mgf=padding.MGF1(hashes.SHA256()),
                    salt_length=padding.PSS.MAX_LENGTH
                ),
                hashes.SHA256()
            )
            # Return 200 status code with message when sign
completed
            return jsonify({'signature': signature.hex()}), 200
```

```
        except Exception as e:
            return jsonify({'error': str(e)}), 500

    else:
        return jsonify({'error': 'Invalid file type'}), 400


# Route for verification
@app.route('/verify', methods=['POST'])
def verify():
    # Check is all necessary data sent(private_key_file, data_file,
signature)
    if 'public_key' not in request.files or 'data_file' not in
request.files or 'signature' not in request.form:
        return jsonify({'error': 'Missing files or signature'}),
400

    # Getting data from request
    public_key_file = request.files['public_key']
    data_file = request.files['data_file']
    signature_hex = request.form['signature']

    # Check if extracted data not empty. Return 400 status code if
empty
    if public_key_file.filename == '' or data_file.filename == ''
or len(signature_hex) == 0:
        return jsonify({'error': 'Empty data'}), 400

    # Check is files have allowed format
    if allowed_file(public_key_file.filename) and
allowed_file(data_file.filename):
        # Get instance of public key from *.pem file
        public_key = load_pem_public_key(
            public_key_file.read(), backend=default_backend()
        )
        data_to_verify = data_file.read()

        try:
            signature = bytes.fromhex(signature_hex)
            # Verify signature
            public_key.verify(
                signature,
                data_to_verify,
                padding.PSS(
                    mgf=padding.MGF1(hashes.SHA256()),
                    salt_length=padding.PSS.MAX_LENGTH
                ),
                hashes.SHA256()
```

```
            )
            return jsonify({'status': 'Signature is valid'})
        except InvalidSignature:
            return jsonify({'status': 'Signature is invalid'})
        except Exception as e:
            return jsonify({'error': str(e)}), 500
    else:
        return jsonify({'error': 'Invalid file types'}), 400


if __name__ == '__main__':
    app.run(debug=True)
```

Процедура підписання документа

Параметри функції підписання:

- data_to_sign: це дані які підписуються. Створений підпис буде унікальним для цих даних. Будь-яка зміна даних призведе до іншого підпису.
- padding: цей параметр визначає схему падінгів, яка буде використовуватися в процесі підписання. Падінги є необхідним у багатьох криптографічних операціях, щоб переконатися, що дані вміщуються в блоки необхідного розміру + додати захист.
    - padding.PSS: розшифровується як Probabilistic Signature Scheme. Це сучасна та безпечна схема доповнення, яка використовується для створення цифрових підписів.
        mgf: функція створення маски. У нашому випадку використовується padding.MGF1(hashes.SHA256()), що означає, що MGF1 із хеш-функцією SHA-256 є функцією генерації маски. MGF1 — це часто використовувана функція створення масок.
        - salt_length: це визначає довжину випадкових дані, що подаються як додатковий вхід.
        - padding.PSS.MAX_LENGTH встановлює довжину випадкових даних на максимально допустиму довжину, що може покращити безпеку підпису.

- hashes.SHA256(): визначає хеш-функцію для використання. Хеш-функція є важливою для цифрових підписів; він стискає дані, які потрібно підписати, до фіксованого розміру, забезпечуючи ефективність і безпеку.

Процес підписання:

- Метод приймає data_to_sign і спочатку хешує його за допомогою SHA-256.
- Потім він шифрує хеш за допомогою закритого ключа за допомогою RSA та вказаної схеми падінгів.
- Результатом є цифровий підпис.

Безпека та використання:

- Безпека цього методу залежить від стійкості приватного ключа, безпеки хеш-функції (SHA-256 у цьому випадку) та ефективності схеми падінгів (PSS з MGF1).

Процес верифікації підпису

На відміну від функції підпису приймає *підпис* та *документ для верифікації* та *публічний ключ.*
Параметри, які відповідають за падінг та хеш-функцію мають співпадати з тими, що були в sign, інакше верифікація не спрацьовує.

```
signature = bytes.fromhex(signature_hex)
            # Verify signature
            public_key.verify(
                signature,
                data_to_verify,
                padding.PSS(
                    mgf=padding.MGF1(hashes.SHA256()),
                    salt_length=padding.PSS.MAX_LENGTH
                ),
                hashes.SHA256()
```

Процес перевірки:

- Метод хешує data_to_verify за допомогою SHA-256.
- Потім він розшифровує підпис за допомогою відкритого ключа та порівнює результат із хешем data_to_verify.

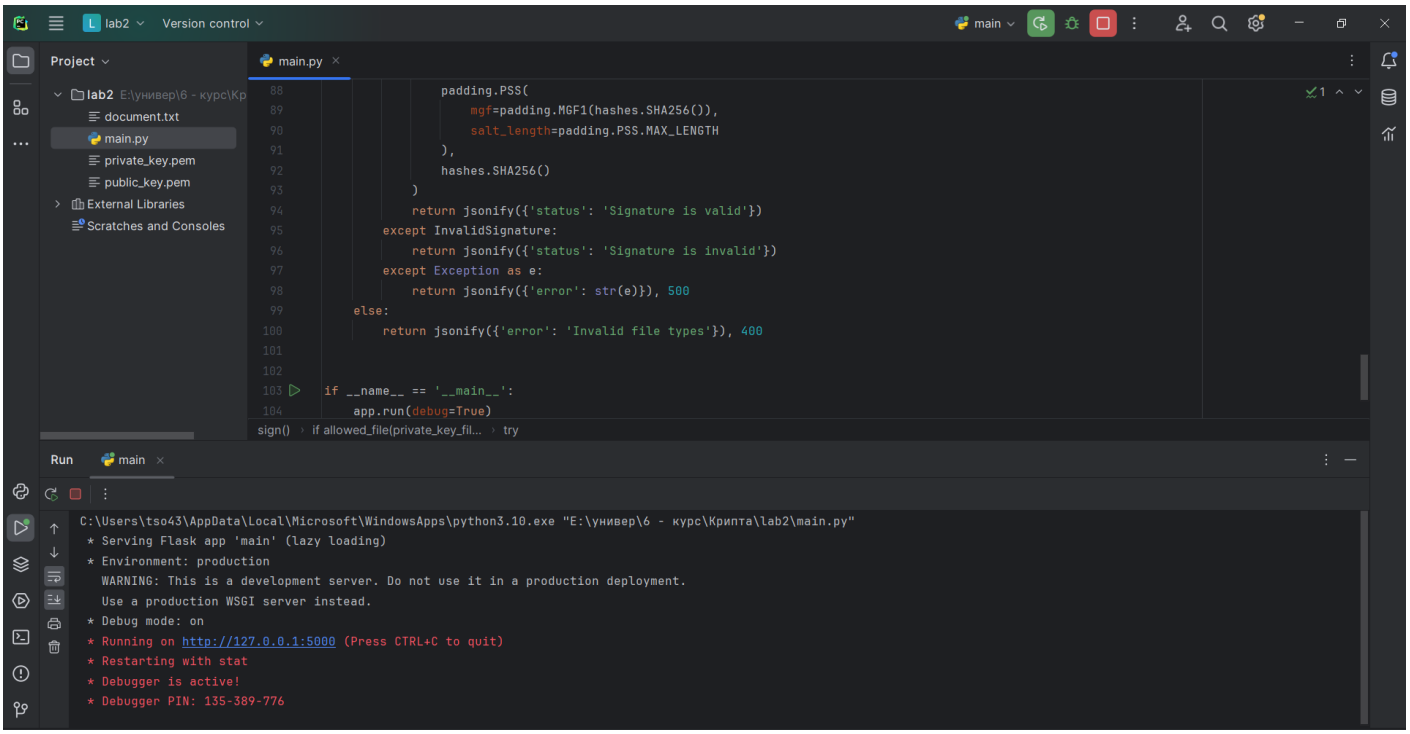
- Якщо розшифрований підпис збігається з хешем, підпис дійсний; інакше виникає виняток, який вказує на те, що перевірка не пройшла.

Безпека та використання:

- Метод перевірки гарантує, що підпис створено власником відповідного закритого ключа та підписані дані не були змінені.
- Він зазвичай використовується в сценаріях перевірки цифрового підпису, такі як: перевірка документів, безпечна передача повідомлень і перевірка особи в криптографічних протоколах.

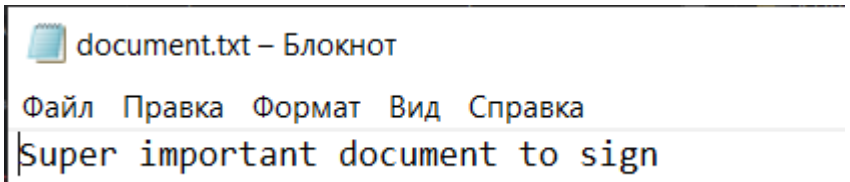
Перевірка роботи сервісу:

Запуск серверу

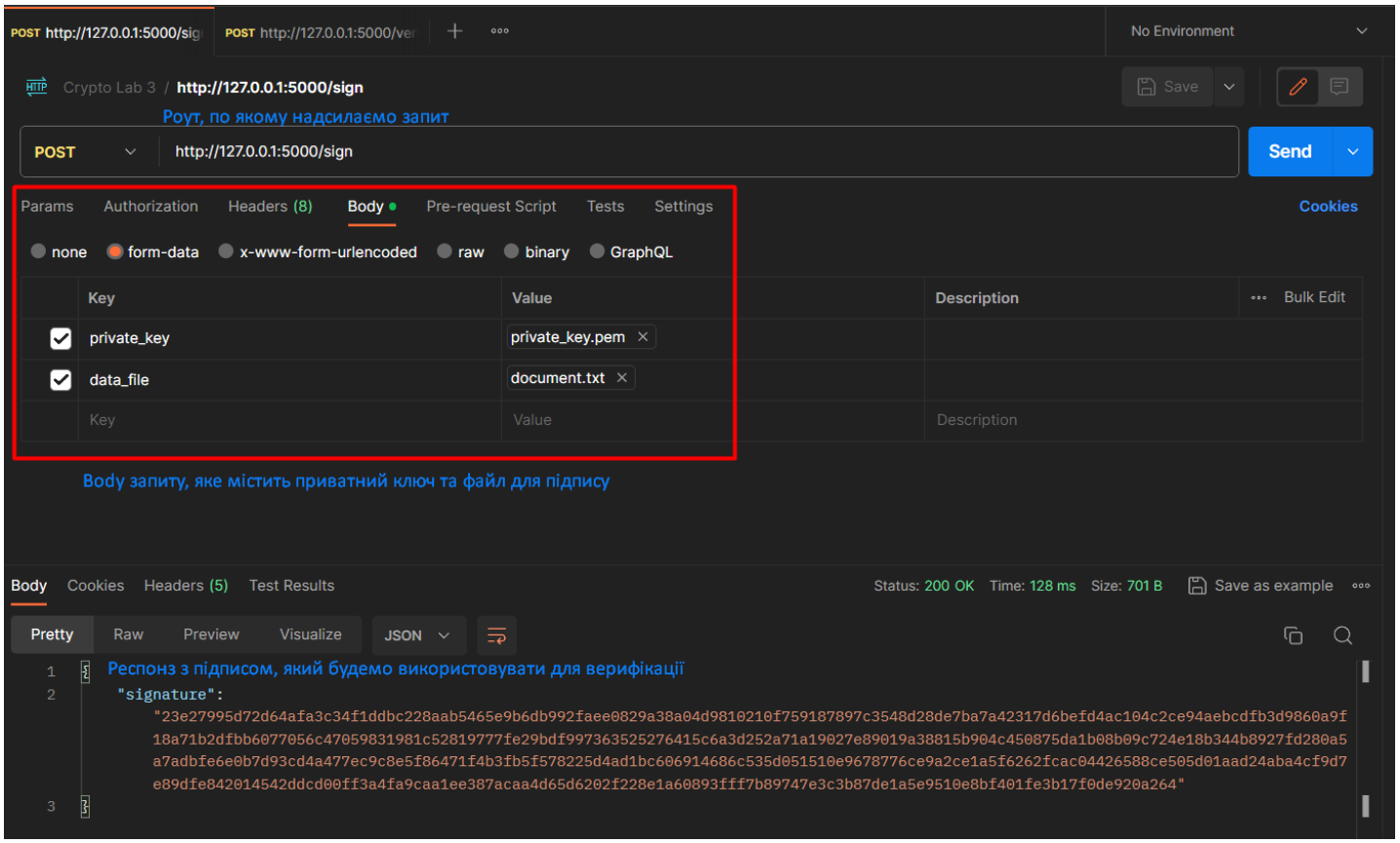


Для надсилання запитів будемо використовувати Postman
Так як надсилати запити в PowerShell ~ 😭🤬
Для початку:

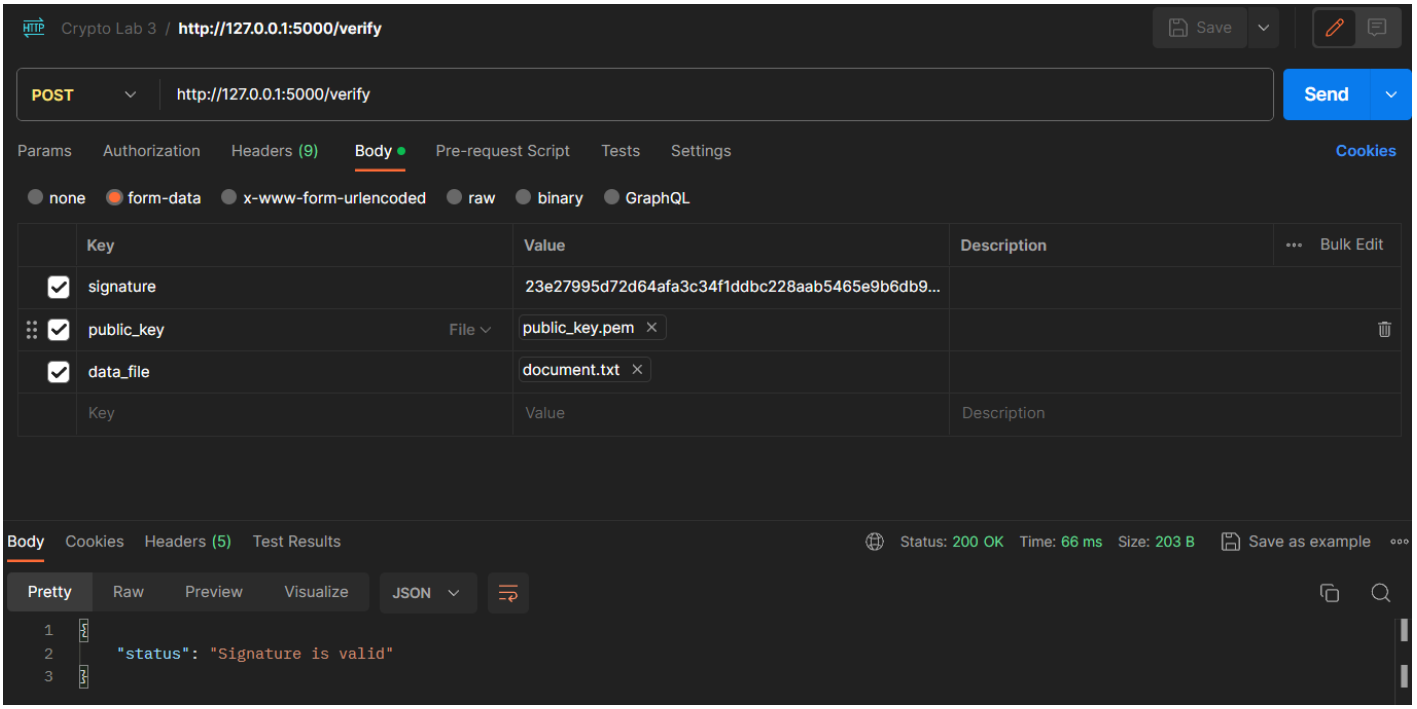
- Документ, який будемо підписувати



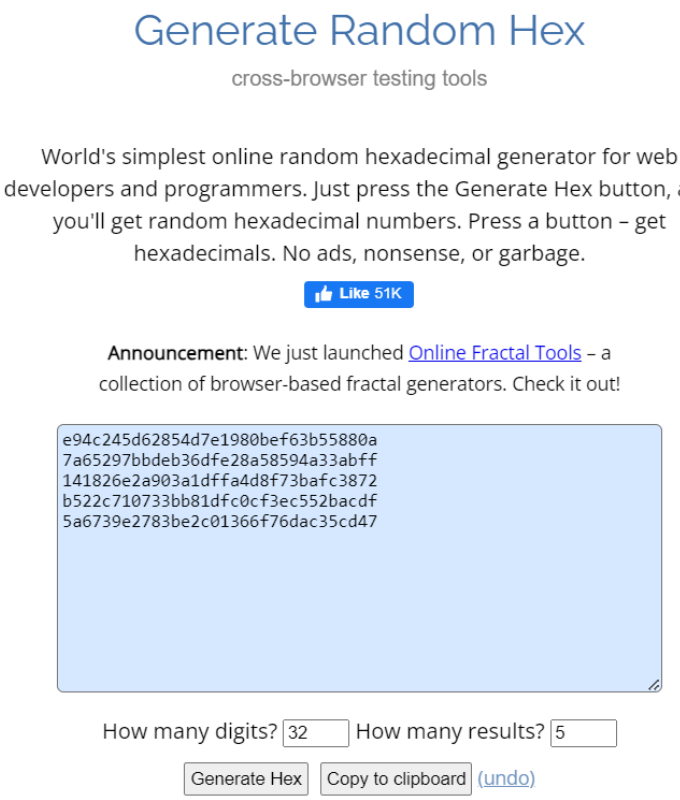
Надсилаємо запит на підписання документу:

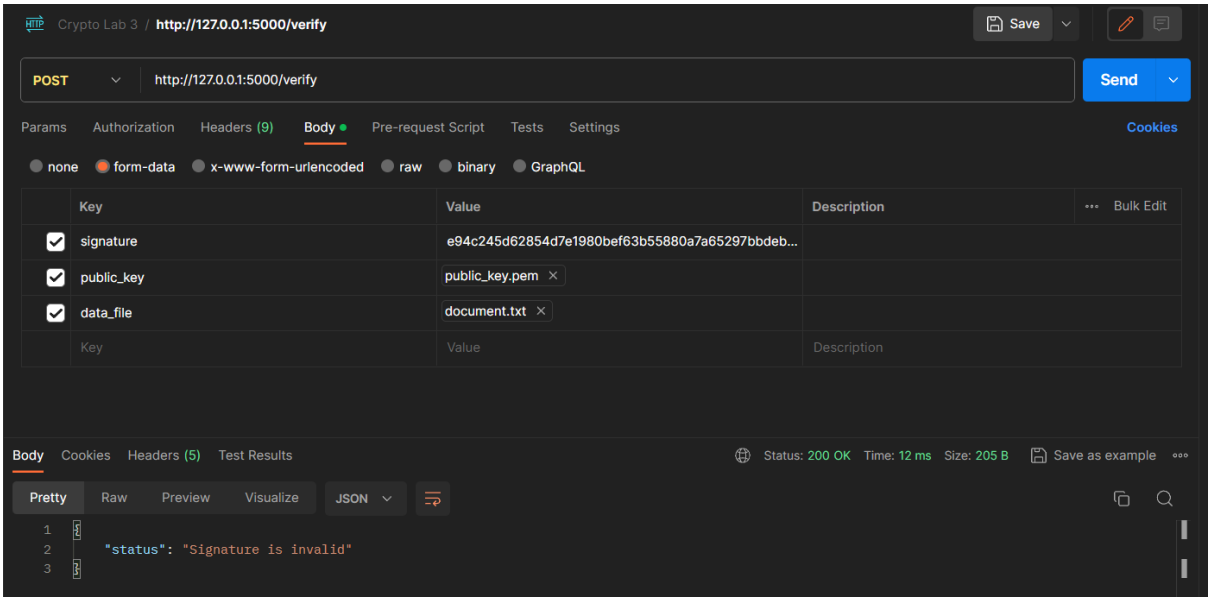


Виконуємо верифікацію підпису:



Некоректний підпис (згенеровано рандомне hex значення):





Висновки:

В результаті роботи було створено сервіс для підпису документів (створення електронного підпису) та верифікації підпису. В сервісі наявні перевірки на коректність параметрів запитів а також хендл помилок, які можуть виникнути під час роботи сервісу для стабільної роботи сервісу. Наведено приклади коректної та не коректної роботи сервісу.