

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
```

[illegible]

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 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 спрацює.

```
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

The screenshot shows a code editor with a Python file named `main.py`. The code defines a Flask application that serves a single route `sign()`. This route takes a file path as input, reads the file, calculates its SHA256 hash, pads it, and signs it using a private key. It then returns a JSON response indicating if the signature is valid or invalid. The script is run in debug mode.

```

88         padding.PSS(
89             mgf=padding.MGF1(hashes.SHA256()),
90             salt_length=padding.PSS.MAX_LENGTH
91         ),
92         hashes.SHA256()
93     )
94     return jsonify({'status': 'Signature is valid'})
95 except InvalidSignature:
96     return jsonify({'status': 'Signature is invalid'})
97 except Exception as e:
98     return jsonify({'error': str(e)}), 500
99 else:
100    return jsonify({'error': 'Invalid file types'}), 400
101
102
103 if __name__ == '__main__':
104    app.run(debug=True)

```

The console output shows the server starting successfully on `http://127.0.0.1:5000`. It includes warnings about the development server and information about the debug mode and debugger.

```

C:\Users\tso43\AppData\Local\Microsoft\WindowsApps\python3.10.exe "E:\универ\6 - курс\Крипта\lab2\main.py"
* Serving Flask app 'main' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 135-389-776

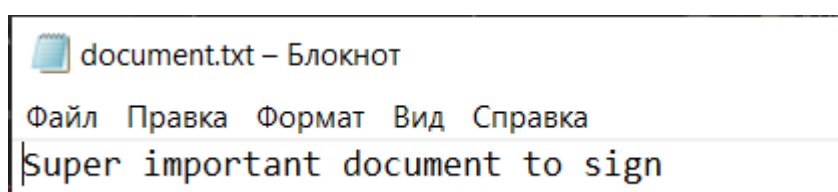
```

Для надсилання запитів будемо використовувати Postman

\*Так як надсилати запити в PowerShell ~ 🤔😡\*

Для початку:

- Документ, який будемо підписувати



Надсилаємо запит на підписання документу:

POST http://127.0.0.1:5000/sign

Crypto Lab 3 / http://127.0.0.1:5000/sign

Route, to which we send the request

POST http://127.0.0.1:5000/sign

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

Key	Value	Description
<input checked="" type="checkbox"/> private_key	private_key.pem	
<input checked="" type="checkbox"/> data_file	document.txt	
Key	Value	Description

Body of the request, which contains the private key and the file for signing

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 128 ms Size: 701 B Save as example

Pretty Raw Preview Visualize JSON

```
1  Респонз з підписом, який будемо використовувати для верифікації
2  "signature":
   "23e27995d72d64afa3c34f1ddbc228aab5465e9b6db992faee0829a38a04d9810210f759187897c3548d28de7ba7a42317d6befd4ac104c2ce94aebcdfb3d9860a9f
   18a71b2dfbb6077056c47059831981c52819777fe29bdf997363525276415c6a3d252a71a19027e89019a38815b904c450875da1b08b09c724e18b344b8927fd280a5
   a7adbfe6e0b7d93cd4a477ec9c8e5f86471f4b3fb5f578225d4ad1bc606914686c535d051510e9678776ce9a2ce1a5f6262fcac04426588ce505d01aad24aba4cf9d7
   e89dfe842014542ddcd00ff3a4fa9caa1ee387acaa4d65d6202f228e1a60893fff7b89747e3c3b87de1a5e9510e8bf401fe3b17f0de920a264"
3
```



Виконуємо верифікацію підпису:

HTTP Crypto Lab 3 / http://127.0.0.1:5000/verify

POST http://127.0.0.1:5000/verify Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	signature	23e27995d72d64afa3c34f1ddbc228aab5465e9b6db9...			
<input checked="" type="checkbox"/>	public_key	File public_key.pem			
<input checked="" type="checkbox"/>	data_file	File document.txt			
	Key	Value	Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 66 ms Size: 203 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": "Signature is valid"
3 }
```

Некоректний підпис (згенеровано рандомне hex значення):

## Generate Random Hex

cross-browser testing tools

World's simplest online random hexadecimal generator for web developers and programmers. Just press the Generate Hex button, and you'll get random hexadecimal numbers. Press a button - get hexadecimals. No ads, nonsense, or garbage.

Like 51K

**Announcement:** We just launched [Online Fractal Tools](#) - a collection of browser-based fractal generators. Check it out!

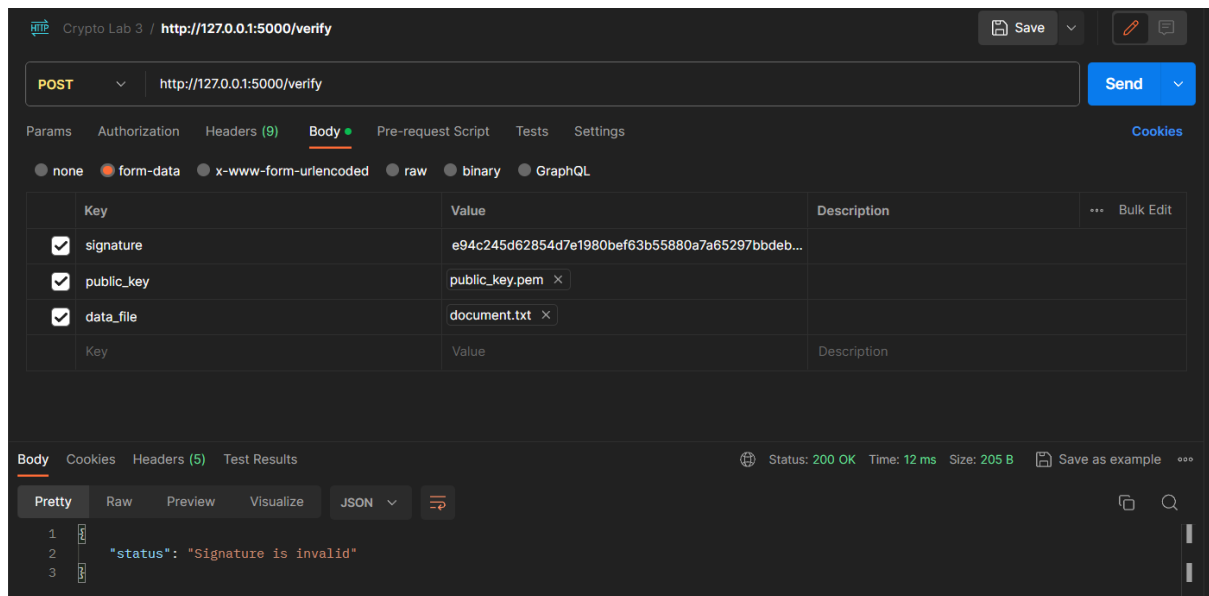
```
e94c245d62854d7e1980bef63b55880a
7a65297bbdeb36dfe28a58594a33abff
141826e2a903a1dffa4d8f73bafc3872
b522c710733bb81dfc0cf3ec552bacdf
5a6739e2783be2c01366f76dac35cd47
```

How many digits?  How many results?

Generate Hex

Copy to clipboard

[undo](#)



## 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.