МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Лабораторна робота №4

З дисципліни «МЕТОДИ РЕАЛІЗАЦІЇ КРИПТОГРАФІЧНИХ МЕХАНІЗМІВ»

Роботу виконали

студенти групи ФІ-21мн, ФТІ

Татенко Вадим

Хмелевський Святослав

Кірсенко Єгор

2023

Мета роботи: Розробка реалізацій Web-сервісу електронного цифрового підпису у відповідності до стандартних вимог Crypto API. (ЗА)

Хід роботи:

Для генерації приватного та публічного ключів було використано бібліотеку OpenSSL:

```
PS E:\универ\6 - курс\Крипта\lab4> openssl genpkey -algorithm RSA -out private_key.pem
>> openssl rsa -pubout -in private_key.pem -out public_key.pem
```

Стандарт вимог Crypto API:

- Всі запити мають бути авторизовані за допомогою апі ключа
- Структура роутів: домен (в нас локалхост), версія продукту, ім’я продукту у spinal-case
- URL in lower case, spinal-case to separate words, query - camelCase
- Максимальна довжина URL 2000 символів
- Структура респонсів:

```
{
 "apiVersion": "",
   "requestId": "",
   "context": "",
   "data": {
      "item": {
           }
   }
}
```

- Хендл можливих помилок

Код сервісу:
Даний код є модифікацією сервісу з лабораторної роботи 3

```
from flask import Flask, request, jsonify, abort
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
from datetime import datetime
```

```
from functools import wraps

app = Flask(__name__)

API_KEY = "your_secret_api_key_here"


def require_api_key(f):
   @wraps(f)
   def decorated_function(*args, **kwargs):
       if request.headers.get('X-API-Key') != API_KEY:
           # Custom JSON payload for unauthorized access
           payload = {
               "apiVersion": "2.0",
               "requestId": "requestId",
               "context": "Authorization Error",
               "data": {
                   "code": 401,
                   "message": "Invalid or missing API key"
               }
           }
           return jsonify(payload), 401
       return f(*args, **kwargs)
   return decorated_function


# 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('/v2/document-processing/sign', methods=['POST'])
@require_api_key
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:
       payload = {
           "apiVersion": "2.0",
           "requestId": "requestId",
           "context": "Sign Error",
           "data": {
               "code": 400,
               "message": "Missing key or data"
           }
       }
       return jsonify(payload), 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 == '':
        payload = {
            "apiVersion": "2.0",
            "requestId": "requestId",
            "context": "Sign Error",
            "data": {
                "code": 400,
                "message": "No selected file"
            }
        }
        return jsonify(payload), 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()
            )
            # Prepare response
            payload = {
                "apiVersion": "2.0",
                "requestId": "requestId",
                "context": "Sign Document",
                "data": {
                    "items": {
                        "signature": signature.hex(),
                        "timestamp":
datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ')
                    }
                }
            }
            # Return 200 status code with message when sign completed
            return jsonify(payload), 200
```

```
        except Exception as e:
            payload = {
                "apiVersion": "2.0",
                "requestId": "requestId",
                "context": "Sign Error",
                "data": {
                    "error": str(e),
                    "code": 400,
                    "message": "Singing Failed"
                }
            }
            return jsonify(payload), 400

    else:
        payload = {
            "apiVersion": "2.0",
            "requestId": "requestId",
            "context": "Sign Error",
            "data": {
                "code": 400,
                "message": "Invalid file type"
            }
        }
        return jsonify(payload), 400


# Route for verification
@app.route('/v2/document-processing/verify', methods=['POST'])
@require_api_key
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:
        payload = {
            "apiVersion": "2.0",
            "requestId": "requestId",
            "context": "Verify error",
            "data": {
                "code": 400,
                "message": "Missing files or signature"
            }
        }
        return jsonify(payload), 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:
        payload = {
            "apiVersion": "2.0",
            "requestId": "requestId",
            "context": "Verify error",
            "data": {
                "code": 400,
                "message": "No data in received files"
            }
        }
        return jsonify(payload), 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()
            )
            payload = {
                "apiVersion": "2.0",
                "requestId": "requestId",
                "context": "Verify success",
                "items": {
                    "status": 'Signature is valid',
                    "timestamp":
datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ')
                }
            }
            return jsonify(payload)
        except InvalidSignature:
            payload = {
                "apiVersion": "2.0",
                "requestId": "requestId",
                "context": "Verify error",
                "data": {
```

```
                    "code": 400,
                    "message": "Signature is invalid"
                }
            }
            return jsonify(payload)
        except Exception as e:
            payload = {
                "apiVersion": "2.0",
                "requestId": "requestId",
                "context": "Unexpected server Error",
                "data": {
                    "error": str(e),
                    "code": 500,
                    "message": "Unexpected server Error"
                }
            }
            return jsonify(payload), 500
    else:
        payload = {
            "apiVersion": "2.0",
            "requestId": "requestId",
            "context": "Verify error",
            "data": {
                "code": 400,
                "message": "Invalid file types"
            }
        }
        return jsonify(payload), 400


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

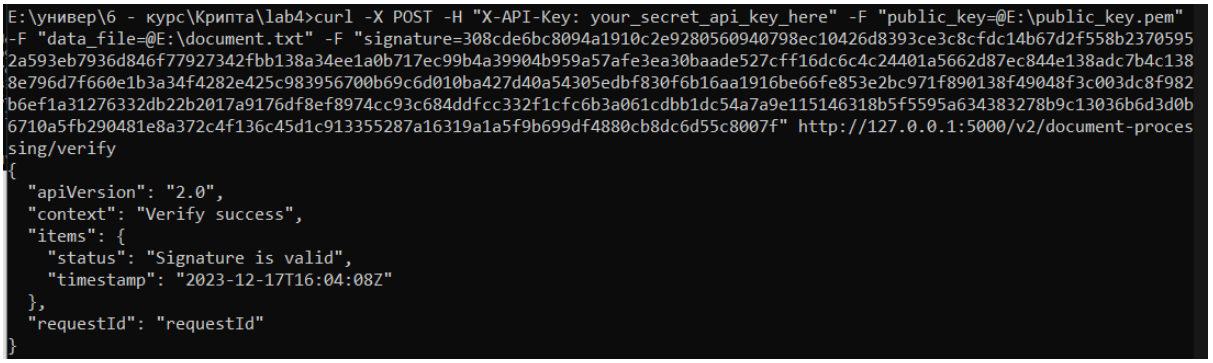
Тестування сервісу:
Надсилаємо запит на підписання документу та отримання підпису:

curl -X POST -H "X-API-Key: your_secret_api_key_here" -F "private_key=@E:\private_key.pem" -F "data_file=@E:\document.txt" http://127.0.0.1:5000/v2/document-processing/sign

```
E:\универ\6 - курс\Крипта\lab4>curl -X POST -H "X-API-Key: your_secret_api_key_here" -F "private_key=@E:\private_key.pem
" -F "data_file=@E:\document.txt" http://127.0.0.1:5000/v2/document-processing/sign
{
  "apiVersion": "2.0",
  "context": "Sign Document",
  "data": {
    "items": {
      "signature": "308cde6bc8094a1910c2e9280560940798ec10426d8393ce3c8cfdc14b67d2f558b23705952a593eb7936d846f77927342fb
b138a34ee1a0b717ec99b4a39904b959a57afe3ea30baade527cff16dc6c4c24401a5662d87ec844e138adc7b4c1388e796d7f660e1b3a34f4282e42
5c983956700b69c6d010ba427d40a54305edbf830f6b16aa1916be66fe853e2bc971f890138f49048f3c003dc8f982b6ef1a31276332db22b2017a91
76df8ef8974cc93c684ddfcc332f1cfc6b3a061cdbb1dc54a7a9e115146318b5f5595a634383278b9c13036b6d3d0b6710a5fb290481e8a372c4f136
c45d1c913355287a16319a1a5f9b699df4880cb8dc6d55c8007f",
      "timestamp": "2023-12-17T16:03:17Z"
    }
  },
  "requestId": "requestId"
```

Надсилаємо запит на верифікацію підпису:

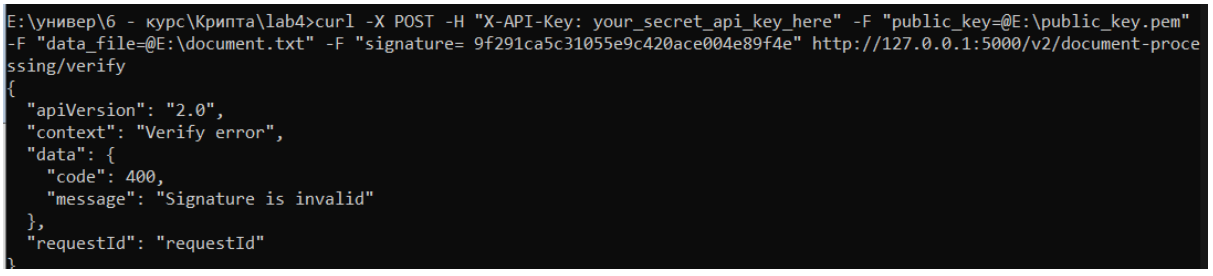
curl -X POST -H "X-API-Key: your_secret_api_key_here" -F "public_key=@E:\public_key.pem" -F "data_file=@E:\document.txt" -F "signature=308cde6bc8094a1910c2e9280560940798ec10426d8393ce3c8cfdc14b67d2f558b23705952a593eb7936d846f77927342fbb138a34ee1a0b717ec99b4a39904b959a57afe3ea30baade527cff16dc6c4c24401a5662d87ec844e138adc7b4c1388e796d7f660e1b3a34f4282e425c983956700b69c6d010ba427d40a54305edbf830f6b16aa1916be66fe853e2bc971f890138f49048f3c003dc8f982b6ef1a31276332db22b2017a9176df8ef8974cc93c684ddfcc332f1cfc6b3a061cdbb1dc54a7a9e115146318b5f5595a634383278b9c13036b6d3d0b6710a5fb290481e8a372c4f136c45d1c913355287a16319a1a5f9b699df4880cb8dc6d55c8007f" http://127.0.0.1:5000/v2/document-processing/verify



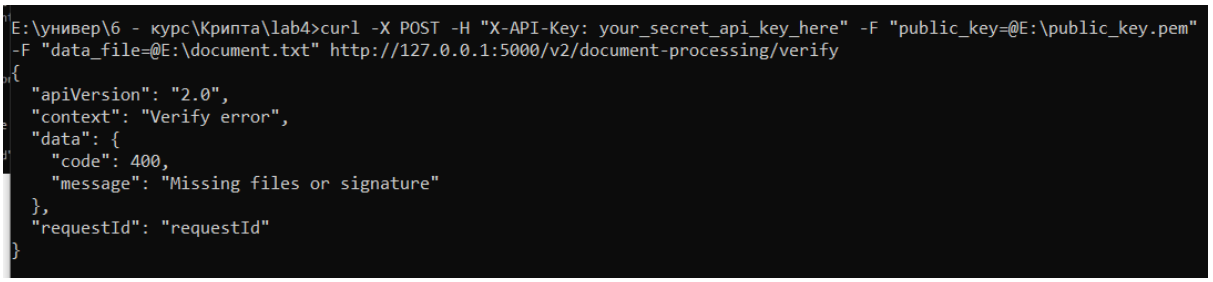
Як бачимо з респонсів сервісу, цей сценарій, коли всі дані передані і опрацьовані коректно.

Приклад некоректних запитів:

Надіслано некоректний підпис



Не надіслано підпис:



Відсутній апі ключ:

```
E:\универ\6 - курс\Крипта\lab4>curl -X POST -F "private_key=@E:\private_key.pem" -F "data_file=@E:\document.txt" http://
127.0.0.1:5000/v2/document-processing/sign
{
  "apiVersion": "2.0",
  "context": "Authorization Error",
  "data": {
    "code": 401,
    "message": "Invalid or missing API key"
  },
  "requestId": "requestId"
}
```

Атака за побічним каналом:

Це тип експлойту безпеки, який не атакує безпосередньо шифрування чи криптографічний алгоритм. Замість цього він використовує слабкі місця в реалізації, прагнучи зібрати інформацію з фізичної реалізації криптосистеми. Поширені атаки за побічним каналах включають атаки за часом, атаки з моніторингом потужності, електромагнітні атаки та акустичний криптоаналіз.

Приклад атаки за часом:

Створимо сценарій, який вимірює час відповіді під час перевірки підписів. Мета зловмисника тут полягає в тому, щоб спостерігати за різницею в часі відповіді, яка може бути спричинена криптографічною обробкою дійсних і недійсних підписів.

```python
import requests
import time
import aiohttp
import asyncio

# Configuration
url_verify = "http://127.0.0.1:5000/v2/document-processing/verify"
public_key_path = "E:/public_key.pem"
data_file_path = "E:/document.txt"
api_key = "your_secret_api_key_here"


# Read files
with open(public_key_path, 'rb') as file:
    public_key = file.read()
with open(data_file_path, 'rb') as file:
    data_file = file.read()

# Sample invalid signature
invalid_signature = "9f291ca5c31055e9c420ace004e89f4e"
```

```
valid_signature =
"58db87ec9bba360925dafa602352450124b8737b13c50a13163249e74f4d66d21
efcee34e8424e563b40e44a89969c53956e2b8b8b34dbfda43f7e376f09a55fea7
e6284bbc3e6f44263cb922033167a5ed439eb36e0a901b7965840d601a14afabef
c265faccd46e84cd47a5a4043d3c337a281075c1f310f5c09714ed4f51ef8ad069
e453452d912c87ccb3f823c1fc33707be3908b7934e144ca0d3200b6f7d977c1c0
23437d956960e89e986f307e1846d7687edb8cdad0f51483f4f1ba0d5c647461e5
0595721e97306b7e0a6ce7967be4a67fedf00288ce0f0946f747c716421c775ed9
66dda36850ce4ca663d32fd5c49cea3b2f2acaaa1bee1043611"


# Function to make a request and measure time
async def measure_response_time(signature):
   async with aiohttp.ClientSession() as session:
       with open(public_key_path, 'rb') as pk_file,
open(data_file_path, 'rb') as data_file:
           data = aiohttp.FormData()
           data.add_field('public_key', pk_file,
filename=public_key_path.split('/')[-1])
           data.add_field('data_file', data_file,
filename=data_file_path.split('/')[-1])
           data.add_field('signature', signature)

           start_time = time.time()
           async with session.post(url_verify, data=data,
headers={'X-API-Key': api_key}) as response:
               elapsed_time = time.time() - start_time
               response_json = await response.json()
               return elapsed_time, response.status, response_json


async def main():
   # time_taken, status, response2_json = await
measure_response_time(valid_signature)
   # print(f"Valid Signature - Time taken: {time_taken} seconds,
Status Code: {status} Response: {response2_json}")
   #
   time_taken, status, response1_json = await
measure_response_time(invalid_signature)
   print(f"InvalidSignature - Time taken: {time_taken} seconds,
Status Code: {status}, Response: {response1_json}")

asyncio.run(main())
```

```
Valid Signature - Time taken: 0.13700222969055176 seconds, Status
 Code: 200 Response: {'apiVersion': '2.0', 'context': 'Verify
 success', 'items': {'status': 'Signature is valid', 'timestamp':
 '2023-12-17T17:35:18Z'}, 'requestId': 'requestId'}
```

```
InvalidSignature - Time taken: 0.15001916885375977 seconds, Status
 Code: 200, Response: {'apiVersion': '2.0', 'context': 'Verify
 error', 'data': {'code': 400, 'message': 'Signature is invalid'},
 'requestId': 'requestId'}
```

Час опрацювання валідного і хибного підпису ~ однаковий. Тому не можна визначити де який.

Висновки:

Під час виконання роботи був створений сервіс електронного підпису документів. Було надано приклади роботи сервісу з прикладами надання некоректних даних під час надсилання запиту. Також Сервіс був створений з виконанням стандартних вимог Crypto API. Після чого було реалізовано один з варіантів атаки за побічним каналом, а саме атака за часом. Результат цієї атаки показав, що сервіс стійкий до такого роду атаки.