

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

**Оформлення результатів роботи.** Контрольний приклад роботи з асиметричною криптосистемою.

## Зміст

Загальні відомості.....	1
Код.....	3
Результат.....	6
Атака за побічним каналом.....	7
Висновки.....	8

## Загальні відомості

Для реалізації було використана віртуальна система VirtualBox 6.1 та віртуальна машина Kali Linux 2023.3. В якості серверу було використано Flask. Також була використана бібліотека cryptography. Також був використаний openssl для створення приватного та публічного ключа.

```
openssl genpkey -algorithm RSA -out private_key.pem
```

```
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

Був створений веб сервіс з мінімальною загрузкою системи. Для створення була використана мова програмування Python 3. В цілому, веб сервіс в цій лабі це модифікація веб сервісу з Лаб 3 але з використанням стандартів Crypto API. В якості джерела інформації даного стандарту була використане наступне джерело

<https://developers.cryptoadis.io/technical-documentation/api/general-information/standards-and-conventions>

Коротко про головні пункти, які повинні задовольняти наш веб сервіс

Вимога до URI

### URI Structure

The Crypto APIs and endpoint URIs follow the RFC 3986 specification and are divided into three main parts, applicable to all Crypto APIs products (Blockchain Events, Blockchain Data, etc.):

1. The domain and subdomain in the URI always display as: **rest.cryptoadis.io**
2. Next is displayed the version of the product, e.g. currently it is: **/v2/**
3. Next is displayed the product name, where words are separated with spinal-case: **/blockchain-data/**

**Example:**

rest.cryptoadis.io/v2/blockchain-data/.

all URLs must be HTTPS - secured, using only HTTP is not accepted.

The entire URI cannot exceed 2000 characters in length including any punctuation marks, e.g. commas, hyphens, question marks, pluses or forward slashes.

Вимога до Post Request:

**POST** requests always contain the following body structure:

```
{
  "apiVersion": "2.0", // The current API version.
  "context": "You can add any text here.", // Optional: In case you send this value in the request
  "data": { // Contains the request parameters.
    "item": { // Single item's details.
      "endpointAttribute": "value" // Parameter data.
    }
  }
}
```

If an endpoint has no parameters, the body structure has less data:

```
{
  "apiVersion": "2.0", // The current API version.
  "context": "You can add any text here", // Optional: In case you send this value in the request
  "data": { // Contains the request parameters.
    "item": { } // Single item's details.
  }
}
```

## Error and HTTP Code Standards

The full list of Errors you can see on our [respective page](#).

The errors we use follow the HTTP Error Codes Standard.

HTTP Status Code	Error it represents
3xx	Redirection Error
4xx	Client Error
5xx	Server Error

Вимога до дат

## Date and Time Formats

For all required endpoints that use a time parameter, the accepted value is `timestamp` which is always UTC-based by definition. The human date which corresponds to the `timestamp` is always converted by the system to UNIX Epoch time and returned as an integer.

### **Example:**

Human time: 09:12:41 AM 28th January 2021

Which corresponds to:

UNIX Epoch time: 1611825161

Hence the system will return `"timestamp": 1611825161` and not the human time.

The UNIX Epoch time is a system that describes a point in time. This would be the amount of time in seconds that have passed since the Unix Epoch which was 00:00:00 UTC on 1 January 1970, minus leap seconds.

Щодо стандартів PKCS, то існує реалізація стандарту PKCS11 (Cryptoki), але через закинутість існуючих реалізацій, повторити його реалізацію є проблематичною задачею в Python.

## Код

```
# Імпорт необхідних модулів з Flask та бібліотеки cryptography
from flask import Flask, request, jsonify
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.exceptions import InvalidSignature
from datetime import datetime

app = Flask(__name__)

@app.route('/v2/blockchain-data/sign', methods=['POST'])
def sign_document():
    # Отримання закритого ключа та документа з запиту
```

```
private_key = request.files['private_key'].read()
document = request.files['document'].read()

# Завантаження закритого ключа для створення цифрового підпису
key = serialization.load_pem_private_key(
    private_key,
    password=None,
    backend=default_backend()
)

# Створення цифрового підпису для документа
signature = key.sign(
    document,
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)

# Підготовка відповіді JSON з цифровим підписом та часовою міткою
response_data = {
    "apiVersion": "2.0",
    "context": "You can add any text here.",
    "data": {
        "item": {
            "signature": signature.hex(),
            "timestamp": datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ')
        }
    }
}
```

```
return jsonify(response_data)
```

```
@app.route('/v2/blockchain-data/verify', methods=['POST'])
```

```
def verify_signature():
```

```
    # Отримання відкритого ключа, документа та цифрового підпису з запиту
```

```
    public_key = request.files['public_key'].read()
```

```
    document = request.files['document'].read()
```

```
    signature = bytes.fromhex(request.form['signature'])
```

```
    # Завантаження відкритого ключа для перевірки цифрового підпису
```

```
    key = serialization.load_pem_public_key(
```

```
        public_key,
```

```
        backend=default_backend()
```

```
)
```

```
try:
```

```
    # Перевірка цифрового підпису документа
```

```
    key.verify(
```

```
        signature,
```

```
        document,
```

```
        padding.PSS(
```

```
            mgf=padding.MGF1(hashes.SHA256()),
```

```
            salt_length=padding.PSS.MAX_LENGTH
```

```
        ),
```

```
        hashes.SHA256()
```

```
)
```

```
    # Підготовка відповіді JSON для валідного підпису
```

```
    response_data = {
```

```
        "apiVersion": "2.0",
```

```
        "requestId": "your_request_id",
```

```
        "context": "",
```

```
        "data": {
```

```

        "item": {"valid": True}
    }
}

return jsonify(response_data)
except InvalidSignature:
    # Підготовка відповіді JSON для невалідного підпису
    response_data = {
        "apiVersion": "2.0",
        "requestId": "your_request_id",
        "context": "",
        "data": {
            "item": {"valid": False}
        }
    }

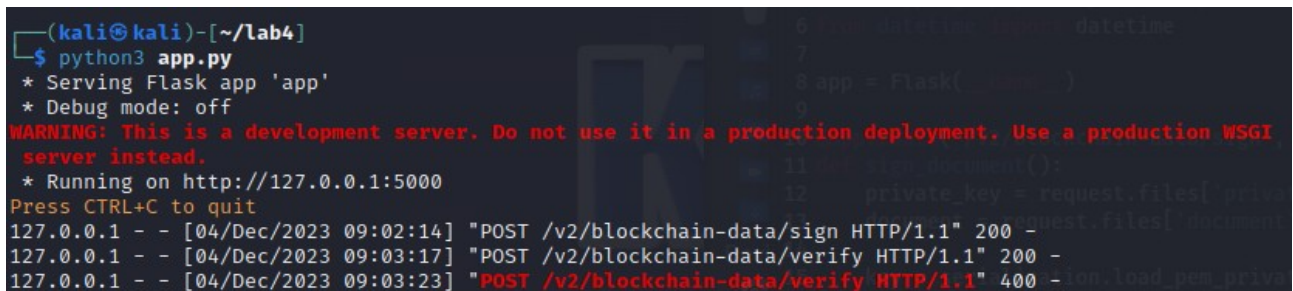
    return jsonify(response_data), 400 # Помилка запиту для невдалої перевірки підпису

if __name__ == '__main__':
    app.run()

```

## Результат

Запуск сервера



```

(kali㉿kali)-[~/lab4]
└─$ python3 app.py
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [04/Dec/2023 09:02:14] "POST /v2/blockchain-data/sign HTTP/1.1" 200 -
127.0.0.1 - - [04/Dec/2023 09:03:17] "POST /v2/blockchain-data/verify HTTP/1.1" 200 -
127.0.0.1 - - [04/Dec/2023 09:03:23] "POST /v2/blockchain-data/verify HTTP/1.1" 400 -

```

Бачимо, що в моменті коли була введена неправильна сигнатура, був код помилки 400, тобто з боку клієнта

Створення сигнатури та перевірка її

```
(kali㉿kali)-[~/lab4]
$ curl -X POST -F "private_key=@private_key.pem" -F "document=@document.txt" http://127.0.0.1:5000//v2/blockchain-data/sign
{"apiVersion":"2.0","context":"You can add any text here.,"data":{"item":{"signature":"68d456af563b53f54fddf3f49eb42c4e0802c1f211715402cccc59091e834b47ed074f742710f5cf18f40c1230aa40aadfbcf0f12dc2e858079ec91bcd87687ba01d26cd31173dbd949e3f818a1bd43f57b45b3faa7960a8fbc33b10892cb843a2aa24e86ef0ceae3c4ea25e96d0b2a1f6d8965a4d91f02933c5f60ae32bec9e2cf20fb3bcd6e56a4ddcf782167ef90c0d00f6e645f710721cc67c59912ec282ca350ced914d57ce71c49fead18d6d6eb9d45c59eef7c0cc99623e134ee9cd055c9602acee97a6eeada33f288cbcbfd559474f24618e79d74125a710a00670a15cdb3821d929a8a6765e4ec0401ee175812f1e1cb46767b36b5a8d946ce240ad","timestamp":"2023-12-04T14:02:14Z"}}}

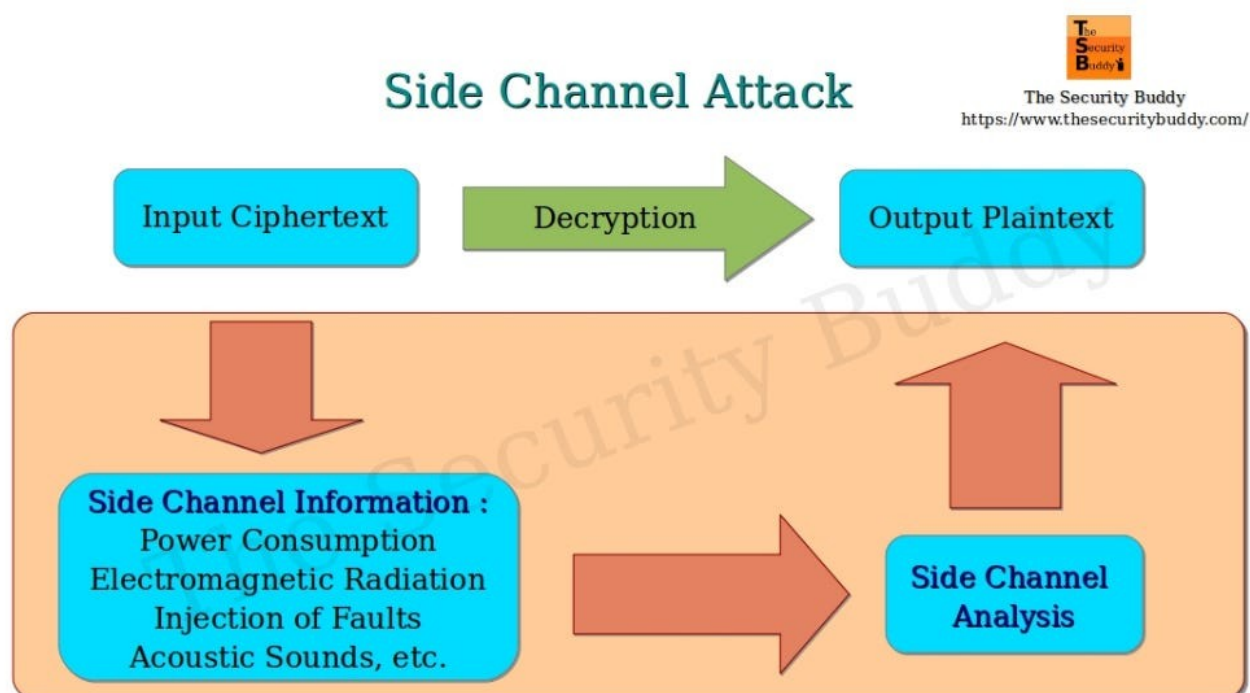
(kali㉿kali)-[~/lab4]
$ curl -X POST -F "public_key=@public_key.pem" -F "document=@document.txt" -F "signature=68d456af563b53f54fddf3f49eb42c4e0802c1f211715402cccc59091e834b47ed074f742710f5cf18f40c1230aa40aadfbcf0f12dc2e858079ec91bcd87687ba01d26cd31173dbd949e3f818a1bd43f57b45b3faa7960a8fbc33b10892cb843a2aa24e86ef0ceae3c4ea25e96d0b2a1f6d8965a4d91f02933c5f60ae32bec9e2cf20fb3bcd6e56a4ddcf782167ef90c0d00f6e645f710721cc67c59912ec282ca350ced914d57ce71c49fead18d6d6eb9d45c59eef7c0cc99623e134ee9cd055c9602acee97a6eeada33f288cbcbfd559474f24618e79d74125a710a00670a15cdb3821d929a8a6765e4ec0401ee175812f1e1cb46767b36b5a8d946ce240ad" http://127.0.0.1:5000/v2/blockchain-data/verify
{"apiVersion":"2.0","context":"","data":{"item":{"valid":true}},"requestId":"your_request_id"}
```

Перевірка неправильної сигнатури

```
(kali㉿kali)-[~/lab4]
$ curl -X POST -F "public_key=@public_key.pem" -F "document=@document.txt" -F "signature=68d456af563b53f54fddf3f49eb42c4e0802c1f211715402cccc59091e834b47ed074f742710f5cf18f40c1230aa40aadfbcf0f12dc2e858079ec91bcd87687ba01d26cd31173dbd949e3f818a1bd43f57b45b3faa7960a8fbc33b10892cb843a2aa24e86ef0ceae3c4ea25e96d0b2a1f6d8965a4d91f02933c5f60ae32bec9e2cf20fb3bcd6e56a4ddcf782167ef90c0d00f6e645f710721cc67c59912ec282ca350ced914d57ce71c49fead18d6d6eb9d45c59eef7c0cc99623e134ee9cd055c9602acee97a6eeada33f288cbcbfd559474f24618e79d74125a710a00670a15cdb3821d929a8a6765e4ec0401ee175812f1e1cb46767b36b5a8d946ce241ad" http://127.0.0.1:5000/v2/blockchain-data/verify
{"apiVersion":"2.0","context":"","data":{"item":{"valid":false}},"requestId":"your_request_id"}
```

## Атака за побічним каналом

В цілому, ось детальний опис даної атаки



Бачимо, що атаку за побічним каналом, можна здійснити за допомогою звуків, електромагнітного випромінювання, тощо. Однак в створеному веб-сервісі такий вид атаки не є коректним, оскільки не використовується графічний інтерфейс, а також використані новітні засоби захисту інформації та криптографічні засоби.

## **Висновки**

В результаті, був створений простий веб сервіс для створення та перевірки електронного підпису. Перевірено правильність виконання коду, а також перевірено можливість створення електронного підпису, а також можливість її перевірки. Також перевірена подія, коли сигнатури не співпадають. Даний веб сервіс був створений за стандартом Crypto API в умовах локального веб сервісу. Також було теоретично перевірена можливість виконання атаки за побічним каналом.