НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО ТЕХНІЧНИЙ ІНСТИТУТ
КАФЕДРА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТА АНАЛІЗУ ДАНИХ

Лабораторна робота №3

З дисципліни «Методи реалізації криптографічних механізмів»

Студента групи ФІ-21мн
Захарченка Л.М.

Викладач:
Селюх П.В.

Київ-2023

ХІД РОБОТИ

1. Завдання

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

2. Результати

```
❯ python3 main.py
INFO:     Started server process [1291449]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     127.0.0.1:44322 - "GET / HTTP/1.1" 404 Not Found
INFO:     127.0.0.1:44322 - "GET /favicon.ico HTTP/1.1" 404 Not Found
INFO:     127.0.0.1:46830 - "POST /generate_keys/ HTTP/1.1" 200 OK
INFO:     127.0.0.1:47574 - "POST /sign/ HTTP/1.1" 200 OK
INFO:     127.0.0.1:33292 - "POST /verify/ HTTP/1.1" 200 OK
```

```
❯ source /home/henitsoi/projects/kpi/mrkm-23-24/venv/bin/activate
❯ curl -X 'POST' \
  'http://127.0.0.1:8000/generate_keys/' \
  -H 'accept: application/json' \
  -d ''

{"private_key_path":"private_key.pem","public_key_path":"public_key.pem"}%

❯ curl -X 'POST' \
  'http://127.0.0.1:8000/sign/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{"message": "This is a secret message"}'

{"signature":"1d4d01c0ed2e9813d2babe701bea0ea5f5177d2b50c3d11a0e7f980b6b9ae16bf607b58bdb4a13734b5430abdc06f73d2207b604b113aee8
7e6c871000b9fb3472ab74cc6e0e145453bf219ed9cc8f873dbc7e5063546c5a8ff82d95b5934913c064a0f2031c6b8d6a0cbc2834460fd6743dd8dfc1ce68
deb243fc1fcf6ad06ca85906b6cb334775d9f81ad8d41fcf2535fe91712201418098d06eb3fa3cc0c7a720634cf27aaca1aa1b4e9554e62d1017d42e6f41f3
d198a84bb51b40e3d80444c40330d609b0a10b4a0ace9fa3beeba62dc2ea202ac77ea3818d4831fc4c9996fc6e247dc0646abae36c345605cc2ff46296b15a
7836f6b18d4d0fce90fa91"}%
❯ curl -X 'POST' \
  'http://127.0.0.1:8000/verify/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "message": "This is a secret message",
  "signature": "1d4d01c0ed2e9813d2babe701bea0ea5f5177d2b50c3d11a0e7f980b6b9ae16bf607b58bdb4a13734b5430abdc06f73d2207b604b113ae
e87e6c871000b9fb3472ab74cc6e0e145453bf219ed9cc8f873dbc7e5063546c5a8ff82d95b5934913c064a0f2031c6b8d6a0cbc2834460fd6743dd8dfc1ce
68deb243fc1fcf6ad06ca85906b6cb334775d9f81ad8d41fcf2535fe91712201418098d06eb3fa3cc0c7a720634cf27aaca1aa1b4e9554e62d1017d42e6f41
f3d198a84bb51b40e3d80444c40330d609b0a10b4a0ace9fa3beeba62dc2ea202ac77ea3818d4831fc4c9996fc6e247dc0646abae36c345605cc2ff46296b1
5a7836f6b18d4d0fce90fa91"
}'
{"valid":true}%
```

Змінив останній символ в зашифрованому хексі підпису:

```
❯ curl -X 'POST' \
  'http://127.0.0.1:8000/verify/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "message": "This is a secret message",
  "signature": "1d4d01c0ed2e9813d2babe701bea0ea5f5177d2b50c3d11a0e7f980b6b9ae16bf607b58bdb4a13734b5430abdc06f73d2207b604b113ae
e87e6c871000b9fb3472ab74cc6e0e145453bf219ed9cc8f873dbc7e5063546c5a8ff82d95b5934913c064a0f2031c6b8d6a0cbc2834460fd6743dd8dfc1ce
68deb243fc1fcf6ad06ca85906b6cb334775d9f81ad8d41fcf2535fe91712201418098d06eb3fa3cc0c7a720634cf27aaca1aa1b4e9554e62d1017d42e6f41
f3d198a84bb51b40e3d80444c40330d609b0a10b4a0ace9fa3beeba62dc2ea202ac77ea3818d4831fc4c9996fc6e247dc0646abae36c345605cc2ff46296b1
5a7836f6b18d4d0fce90fa92"
}'
{"valid":false}%
```

3. Код

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes
from cryptography.exceptions import InvalidSignature

app = FastAPI()

BITS = 2048
PRIVATE_KEY_PATH = "private_key.pem"
PUBLIC_KEY_PATH = "public_key.pem"

class Message(BaseModel):
    message: str


class Signature(BaseModel):
    message: str
    signature: str


def load_private_key(private_key_path):
    with open(private_key_path, "rb") as key_file:
        private_key = serialization.load_pem_private_key(
            key_file.read(),
            password=None,
        )
    return private_key


def load_public_key(public_key_path):
    with open(public_key_path, "rb") as key_file:
        public_key = serialization.load_pem_public_key(
            key_file.read(),
        )
    return public_key


# Generate RSA keys and save to files
@app.post("/generate_keys/")
```

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

    with open(PRIVATE_KEY_PATH, "wb") as f:
        f.write(private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.PKCS8,
            encryption_algorithm=serialization.NoEncryption()
        ))

    with open(PUBLIC_KEY_PATH, "wb") as f:
        f.write(public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        ))

    return {
        "private_key_path": PRIVATE_KEY_PATH,
        "public_key_path": PUBLIC_KEY_PATH
    }


# Sign a message
@app.post("/sign/")
def sign_message(message: Message):
    private_key = load_private_key(PRIVATE_KEY_PATH)
    message_bytes = message.message.encode()
    signature = private_key.sign(
        message_bytes,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    return {"signature": signature.hex()}


# Verify a signature
@app.post("/verify/")
def verify_signature(signature: Signature):
```

```
    public_key = load_public_key(PUBLIC_KEY_PATH)
    message_bytes = signature.message.encode()
    signature_bytes = bytes.fromhex(signature.signature)
    try:
        public_key.verify(
            signature_bytes,
            message_bytes,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )
        return {"valid": True}
    except InvalidSignature:
        return {"valid": False}
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))


if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```