НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Кафедра Інформаційної Безпеки

Лабораторна робота №3

з дисципліни

«МЕТОДИ РЕАЛІЗАЦІЇ КРИПТОГРАФІЧНИХ МЕХАНІЗМІВ»

Виконали:

Студени 6 курсу ФТІ

групи ФБ-41мн

Бондаренко О.Ю., Кригін Д.О.

Перевірила:

асистент

Байденко П.В.

Київ 2025

Лабораторна робота №3
Реалізація основних асиметричних криптосистем.

Мета: Дослідження можливостей побудови загальних та спеціальних криптографічних протоколів за допомогою асиметричних криптосистем.

Завдання: Другий тип лабораторної роботи. Розробити реалізацію асиметричної криптосистеми. Підгрупа 2В. Бібліотека PyCrypto під Linux платформу. Стандарт ECDSA.

Хід Роботи

Теоретичні відомості

P-256 (NIST P-256)

Роль у коді: Використовується у функції ECC.generate(curve="P-256") для визначення параметрів генерації пари ключів.

Особливості:

Це специфічна стандартизована еліптична крива, визначена Національним інститутом стандартів і технологій США (NIST).

Цифра "256" вказує на довжину ключа (256 біт), що визначає її криптографічну стійкість.

Забезпечує 128-бітний рівень безпеки, що за стійкістю є еквівалентом приблизно 3072-бітного ключа RSA.

Її стандартизація забезпечує високу сумісність між різними криптографічними системами та бібліотеками.

SHA-256 (Secure Hash Algorithm 256)

Роль у коді: Використовується у SHA256.new(message) для створення гешу повідомлення перед його підписанням або перевіркою.

Особливості:

Швидкий та надійний геш-алгоритм, який є стандартом в індустрії.

fips-186-3 (Digital Signature Standard)

Роль у коді: Вказується як режим у DSS.new(key, "fips-186-3").

Особливості:

Визначає правила та формати, за якими має відбуватися процес створення та перевірки цифрового підпису.

У коді він вказує бібліотеці pycryptodome, що для роботи з наданим ECC-ключем (P-256) та гешем (SHA-256) слід використовувати алгоритм ECDSA (Digital Signature Algorithm на еліптичних кривих) точно так, як це описано у стандарті FIPS 186-3.

Він стандартизує математичні операції, які виконують методи .sign() та .verify(), забезпечуючи сумісність підписів, створених цією бібліотекою, з іншими системами, що дотримуються того ж стандарту.

Формат PEM

Роль у коді: Використовується для зберігання та передачі ключів у текстовому вигляді (.export_key(format="PEM") та .import_key(private_key_pem)).

Особливості:

PEM (Privacy-Enhanced Mail) — це текстовий формат кодування бінарних даних, найчастіше криптографічних ключів та сертифікатів. Використовує кодування base64 для текстового представлення даних.

Це дозволяє легко копіювати ключі (наприклад, copy-paste в email або текстовий файл) без ризику пошкодити їх.

Файл у форматі PEM легко впізнати, оскільки він починається та закінчується специфічними заголовками, наприклад:

-----BEGIN PUBLIC KEY-----

(Тут дані у кодуванні Base64)

-----END PUBLIC KEY-----

У наданому коді саме в цьому форматі ключі зберігаються у файлах privkey.pem та pubkey.pem.

Реалізація

В якості асиметричної криптосистеми, був реалізований веб сервіс для накладання та перевірки цифрового підпису алгоритмом ECDSA.

1. Генерація Ключів (GET /generate_keys)

Створення нової пари ключів (приватного та публічного) для асиметричного шифрування. Використовується модуль Crypto.PublicKey.ECC для генерації

пари ключів (ECC.generate). Як параметр вказується конкретна еліптична крива: P-256. Приватний та публічний ключі експортуються у текстовий формат PEM. Обидва .pem файли (privkey.pem та pubkey.pem) запаковуються у ZIP-архів (keys.zip). Архів повертається користувачеві для завантаження.

2. Створення Підпису (POST /sign)

Підписання файлу (повідомлення) за допомогою приватного ключа. Сервер очікує завантаження двох файлів: приватного ключа (privkey) та файлу повідомлення (message). Повідомлення зчитується та гешується за допомогою алгоритму SHA-256. Приватний ключ імпортується з формату PEM. Створюється об'єкт підпису (DSS.new) з використанням ключа та специфікації fips-186-3. Метод .sign() об'єкта DSS використовується для створення цифрового підпису на основі гешу повідомлення (а не самого повідомлення). Бінарний підпис повертається користувачеві як файл signature.bin.

3. Перевірка Підпису (POST /verify)

Верифікація автентичності та цілісності повідомлення за допомогою публічного ключа та підпису. Сервер очікує завантаження трьох файлів: публічного ключа (pubkey), оригінального повідомлення (message) та файлу підпису (signature). Повідомлення зчитується та гешується (знову) за допомогою SHA-256. Створюється об'єкт верифікатора (DSS.new) за стандартом fips-186-3. Метод .verify() порівнює наданий підпис з гешем повідомлення, використовуючи математику публічного ключа. Сервер повертає JSON-відповідь:

{"verified": true} — якщо підпис дійсний.

{"verified": false, "error": "Invalid signature"} - якщо підпис недійсний або пошкоджений.

Демонстрація роботи

Запуск сервера

```
[main][kali: ~/mrkm-2025/lab3]$ ./setup.sh
Creating virtualenv lab3 in /home/user/mrkm-2025/lab3/.venv
Installing dependencies from lock file

Package operations: 10 installs, 0 updates, 0 removals

  - Installing markupsafe (3.0.3)
  - Installing blinker (1.9.0)
  - Installing click (8.3.0)
  - Installing itsdangerous (2.2.0)
  - Installing jinja2 (3.1.6)
  - Installing packaging (25.0)
  - Installing werkzeug (3.1.3)
  - Installing pycryptodome (3.23.0)
  - Installing flask (3.1.2)
  - Installing gunicorn (23.0.0)
[main][kali: ~/mrkm-2025/lab3]$ ./run.sh
[2025-11-14 11:49:48 +0200] [7607] [INFO] Starting gunicorn 23.0.0
[2025-11-14 11:49:48 +0200] [7607] [INFO] Listening at: http://0.0.0.0:5000 (7607)
[2025-11-14 11:49:48 +0200] [7607] [INFO] Using worker: sync
[2025-11-14 11:49:48 +0200] [7610] [INFO] Booting worker with pid: 7610
```

Генерація пари ключів

```
[kali: ~/Desktop]$ curl -s http://127.0.0.1:5000/generate_keys -o keys.zip
[kali: ~/Desktop]$ unzip keys.zip
Archive:  keys.zip
  inflating: privkey.pem
  inflating: pubkey.pem
[kali: ~/Desktop]$ cat privkey.pem
-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgdd3pYRnqM20Mo2O7
NATdIiHJJOFL9FiFdNdBzGAyAWyhRANCAAQwgAMn/egti1UYUXnp9/Cn8sFSWLn3
SHeR5hhhJ2xyLX1gPuLm+omQDxaqma4Pz1KuoCUm/m4qFaso81JCwEOS
-----END PRIVATE KEY-----%
[kali: ~/Desktop]$ cat pubkey.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEMIADJ/3oLYtVGFF56ffwp/LBUli5
90h3keYYYSdsci19YD7i5vqJkA8WqpmuD89SrqAlJv5uKhWrKPNSQsBDkg==
-----END PUBLIC KEY-----%
[kali: ~/Desktop]$
```

Підписання повідомлення

```
[kali: ~/Desktop]$ echo "This is the message to sign" > message.txt
[kali: ~/Desktop]$ curl -s -X POST http://127.0.0.1:5000/sign \
     -F "privkey=@privkey.pem" \
     -F "message=@message.txt" \
     -o signature.bin
[kali: ~/Desktop]$ xxd signature.bin
00000000: 655d e15c d512 6e5c fd02 f05c 074e 7589  e].\..n\...\.Nu.
00000010: 8b9b 316f c252 0d11 e6d0 a4ee e993 f794  ..1o.R..........
00000020: c375 ae44 d086 7ecc f924 005d 7f0c 7ad2  .u.D..~..$.]..z.
00000030: 19ae af22 36f5 8947 6894 edf2 d5ad cfd3  ..."6..Gh.......
[kali: ~/Desktop]$
```

Перевірка підпису повідомлення (успішна)

```
[kali: ~/Desktop]$ curl -X POST http://127.0.0.1:5000/verify \
     -F "pubkey=@pubkey.pem" \
     -F "message=@message.txt" \
     -F "signature=@signature.bin"
{"verified":true}
[kali: ~/Desktop]$
```

Перевірка підпису повідомлення (хибна)

```
[kali: ~/Desktop]$ echo 'FAKE SIGNATURE' > signature.bin
[kali: ~/Desktop]$ curl -X POST http://127.0.0.1:5000/verify \
     -F "pubkey=@pubkey.pem" \
     -F "message=@message.txt" \
     -F "signature=@signature.bin"
{"error":"Invalid signature","verified":false}
[kali: ~/Desktop]$
```

Висновки

Відповідно до індивідуального завдання, було розроблено та реалізовано веб-сервіс, що демонструє повний цикл роботи алгоритму цифрового підпису ECDSA. Цей сервіс успішно виконує три ключові функції: генерує пари ключів на стандартизованій еліптичній кривій P-256, створює цифровий підпис для наданого повідомлення за допомогою приватного ключа та верифікує підпис, використовуючи публічний ключ.