НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Звіт з виконання лабораторної роботи

ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ РЕАЛІЗАЦІЇ ІСНУЮЧИХ ПРОГРАМНИХ СИСТЕМ ЯКІ ВИКОРИСТОВУЮТЬ КРИПТОГРАФІЧНІ МЕХАНІЗМИ ЗАХИСТУ ІНФОРМАЦІЇ

Виконали студенти
групи ФІ-52мн
Волошин Ігор
Геращенко Володимир
Дорошенко Юрій

Варіант 2Б

Київ — 2025

1 Мета роботи

розробити реалізацію асиметричної криптосистеми у відповідності до стандартних вимог Crypto API або стандартів PKCS та дослідити стійкість стандартних криптопровайдерів до атак, що використовують недосконаліть механізмів захисту операційної системи.

2 Опис використаних стандартів та інтерфейсів прикладного програмування (API)

2.1 Стандарти та специфікації

У розробленій програмній реалізації використано набір міжнародних криптографічних стандартів, що забезпечують сумісність, безпеку та валідність цифрових підписів.

- **Алгоритм підпису: ECDSA (Elliptic Curve Digital Signature Algorithm).** Реалізовано згідно зі стандартом **FIPS 186-4** (Digital Signature Standard). ECDSA забезпечує той самий рівень безпеки, що й RSA, але при значно меншій довжині ключа, що зменшує навантаження на обчислювальні ресурси.
- **Еліптична крива: NIST P-256 (secp256r1).** Використано криву над простим полем, визначену стандартом FIPS 186. Довжина ключа становить 256 біт, що забезпечує рівень стійкості, еквівалентний приблизно 128 бітам симетричного шифрування або 3072 бітам алгоритму RSA.
- **Функція хешування: SHA-256.** Використовується для створення дайджесту повідомлення перед підписанням, згідно зі стандартом **FIPS 180-4** (Secure Hash Standard).
- **Формат зберігання ключів: PKCS #8.** Приватні ключі експортуються та зберігаються у форматі PEM (Privacy-Enhanced Mail), що базується на стандарті синтаксису інформації про приватні ключі (Private-Key Information Syntax Standard). Публічні ключі зберігаються у форматі X.509 (SubjectPublicKeyInfo).
- **Детермінований підпис (RFC 6979).** Для генерації одноразового числа k (nonce) використовується детермінований алгоритм на основі HMAC-DRBG, що усуває ризик витоку приватного ключа через неякісний генератор випадкових чисел (RNG).

2.2 Інтерфейс методів бібліотеки PyCrypto (PyCryptodome)

Програмна реалізація базується на об'єктно-орієнтованому API бібліотеки `PyCryptodome`, яка є сучасним форком `PyCrypto`. Нижче наведено опис основних класів та методів, що були задіяні в лабораторній роботі.

2.2.1 Модуль `Crypto.PublicKey.ECC`

Відповідає за операції з еліптичними кривими, генерацію та серіалізацію ключів.

`ECC.generate(curve='P-256')`
: Статичний метод, що генерує нову пару ключів (публічний та приватний) на основі заданої кривої. Повертає об'єкт ключа `EccKey`.

`key.export_key(format='PEM')`
: Експортує об'єкт ключа у текстовий рядок байтів. Формат 'PEM' додає стандартні заголовки `---BEGIN PRIVATE KEY---` та кодує тіло ключа в Base64.

`ECC.import_key(data)`
: Відновлює об'єкт `EccKey` із зовнішнього представлення (рядка або файлу). Автоматично визначає тип ключа (публічний/приватний) та формат кодування.

2.2.2 Модуль `Crypto.Signature.DSS`

Реалізує стандарт цифрового підпису (Digital Signature Standard).

`DSS.new(key, mode='fips-186-3')`
: Створює об'єкт підписувача (якщо передано приватний ключ) або веріфікатора (якщо передано публічний).

  - `key`: об'єкт `EccKey`.
  - `mode`: режим генерації k. Значення `'fips-186-3'` вказує на використання детермінованого підходу для запобігання атакам на RNG.

`signer.sign(hash_object)`
: Приймає об'єкт хешу (SHA-256) та повертає байтовий рядок підпису (конкатенація чисел r та s).

`verifier.verify(hash_object, signature)`
: Перевіряє відповідність підпису хешу повідомлення. Якщо підпис недійсний, метод викликає виключення `ValueError`. Не повертає значення (`None`) у разі успіху.

2.2.3 Модуль `Crypto.Hash.SHA256`

Забезпечує створення криптографічних дайджестів.

`SHA256.new(data=None)`
: Створює новий об'єкт хешування. Може приймати дані для хешування при ініціалізації.

`hash.hexdigest()`
: Повертає рядкове представлення хешу у шістнадцятковому форматі (використовується для логування та візуальної перевірки).

3 Дослідження стійкості до атак на оперативну пам'ять та механізми підкачки (Swap)

3.1 Формальна постановка проблеми (Data Remanence)

Атака на залишкову інформацію (Data Remanence) базується на властивості фізичних носіїв та операційних систем зберігати дані після завершення їх активного використання.

Нехай K_{priv} — приватний ключ асиметричної криптосистеми. Нехай P — процес, що виконує криптографічні перетворення. Множина адресного простору процесу M_P у момент часу t містить ключ:

$$K_{priv} \in M_P(t)$$

Після виконання операції підпису $S = Sign(K_{priv}, m)$, прикладна логіка програми може втратити посилання на об'єкт ключа. Однак, у середовищах з автоматичним керуванням пам'яттю (Python, Java), фізичне видалення даних з комірок пам'яті RAM не відбувається

миттєво. Час існування копії ключа t_{exist} визначається роботою збирача сміття (Garbage Collector) та аллокатора пам'яті ОС:

$$t_{exist} > t_{usage}$$

Зловмисник A, що має права доступу до системи (локальний користувач або root), може здійснити атаку типу *Cold Boot* або *Memory Dump* у проміжок часу $\Delta t = t_{exist} - t_{usage}$.

3.2 Вектори реалізації атаки в середовищі Linux

У ході роботи було змодельовано два вектори атак на реалізовану систему ECDSA.

3.2.1 Атака через файл підкачки (Swap Paging Attack)

Операційна система Linux використовує механізм віртуальної пам'яті. При нестачі фізичної RAM, сторінки пам'яті (pages) неактивних процесів вивантажуються на жорсткий диск у розділ або файл підкачки (Swap). Якщо процес P, що зберігає K_{priv}, переходить у режим очікування (наприклад, 'input()'), ОС може виконати операцію *swap-out*.

Оскільки Swap є енергонезалежним сховищем, K_{priv} зберігається на диску у відкритому вигляді (якщо не налаштовано шифрування Swap). Зловмисник може проаналізувати вміст блокового пристрою:

```
$ strings /dev/dm-1 | grep "BEGIN PRIVATE KEY" -A 5
```

3.2.2 Атака через дамп пам'яті (Live Process Dump)

Використовуючи системний виклик `ptrace` (доступний через утиліти налагодження, такі як `gdb`), можна отримати повний зліпок віртуальної пам'яті процесу без його зупинки.

Демонстрація атаки:

1. Отримання PID процесу жертви: `pgrep -f ecdsa_lab.py`.
2. Створення дампу пам'яті у файл 'core':

   ```
   $ sudo gcore <PID>
   ```

3. Екстракція ключа з бінарного файлу дампу:

   ```
   $ strings core.<PID> | grep -A 3 "MIGHAgEAMBM"
   ```

   (де `MIGH...` — сигнатура заголовка ключа у форматі Base64/DER).

Результат експерименту підтвердив, що стандартні змінні типу `bytes` у Python зберігають криптографічний матеріал у відкритому вигляді до моменту перезапису пам'яті іншими даними або завершення процесу.

Можна побачити другу атаку у дії у рисунку 1.

3.3 Методи протидії та захисту

Для побудови захищених гібридних криптосистем необхідно реалізувати комплекс заходів на рівні ОС та коду програми.

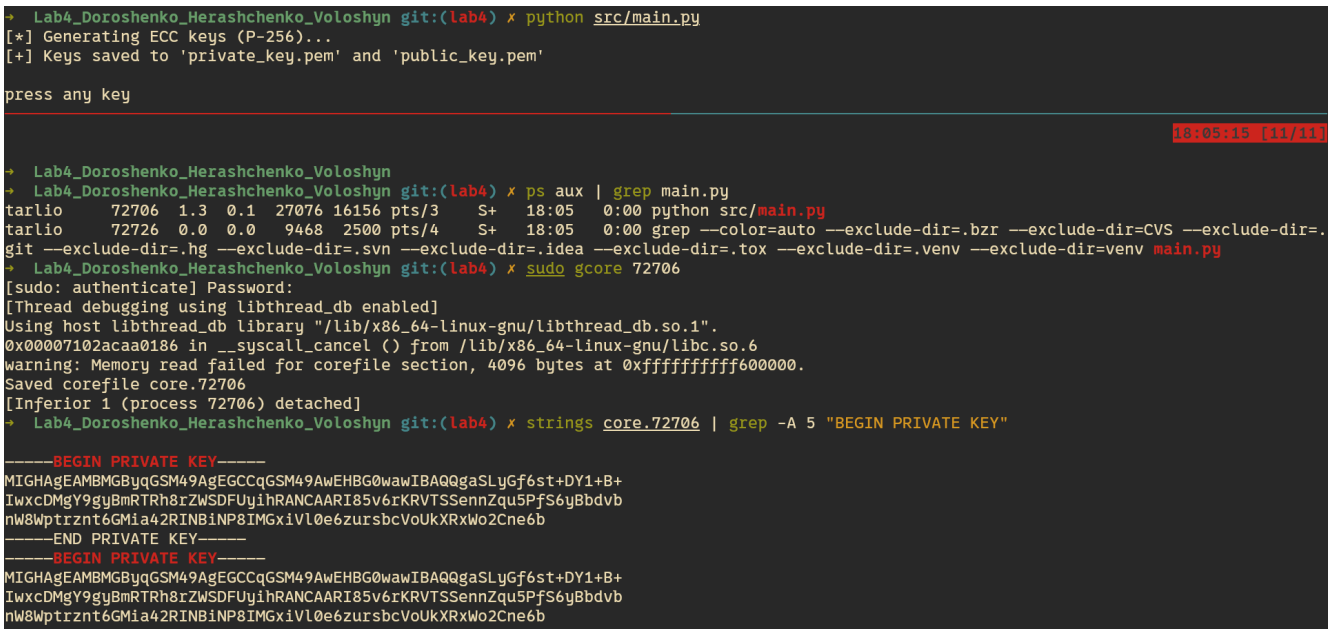

Рис. 1: Атака через дамп пам'яті

3.3.1 Захист на рівні операційної системи

– **Вимкнення Swap:** Для критичних серверів рекомендується повне відключення підкачки (`swapoff -a`) або шифрування розділу підкачки (LUKS Encrypted Swap), що робить вилучені дані ентропійним шумом для зловмисника.

– **Обмеження ptrace:** Встановлення параметра ядра `kernel.yama.ptrace_scope = 1` (або 2/3) забороняє непривілейованим користувачам підключатися до процесів для зчитування пам'яті.

3.3.2 Захист на рівні програмної реалізації (Secure Coding)

Основною проблемою Python є незмінність (immutability) типів `str` та `bytes`. При спробі зміни змінної створюється її копія, а оригінал залишається в пам'яті.

Рекомендовані техніки:

1. **Використання змінних буферів (Mutable Buffers):** Замість `bytes` слід використовувати `bytearray`. Це дозволяє виконувати "затирання"(zeroization) секретних даних після використання:

```
key_buffer = bytearray(b'secret_key_material')
# ... usage ...
# Secure wipe
for i in range(len(key_buffer)):
    key_buffer[i] = 0
```

2. **Блокування сторінок пам'яті (Memory Locking):** Використання системного виклику `mlock` через модуль `ctypes` дозволяє заборонити ОС вивантажувати конкретну область пам'яті у Swap.

3. **Мінімізація часу життя секрету:** Ключ має розшифровуватися або завантажуватися в пам'ять лише на момент виконання операції підпису і негайно видалятися (перезаписуватися).

4 Висновок

Забезпечення конфіденційності в асиметричних криптосистемах вимагає виходу за межі математичних моделей і врахування особливостей керування пам'яттю конкретної програмно-апаратної платформи.