



Міністерство освіти і науки України  
Національний технічний університет України  
“Київський політехнічний інститут імені Ігоря Сікорського”  
Навчально-науковий фізико технічний інститут  
Кафедра інформаційної безпеки

## **Звіт**

З практичного завдання №4

із дисципліни «Методи реалізації криптографічних механізмів»

**Тема: «Дослідження особливостей реалізації існуючих програмних систем,  
які використовують криптографічні механізми захисту інформації»**

Виконав:  
Студент групи ФБ-41мн  
Шерстюк А. В.  
Виграновський М.В

**Київ-2025**

# Реалізація асиметричної криптосистеми під Linux платформу. Стандарт ECDSA

Дослідження стійкості криптопровайдера до атак, що виникають через недоліки захисту операційної системи, показує: навіть бездоганна з математичного погляду реалізація ECDSA не рятує, якщо саме оточення скомпрометоване. Надійність криптопровайдера залежить не лише від математичної моделі, але й від цілого комплексу умов, що включають безпечне зберігання ключів та налаштування доступу в ОС.

Загрози та вразливості криптопровайдера:

- 1. Викрадення приватного ключа:** Найочевидніша загроза- крадіжка приватного ключа з файлової системи. Якщо файл, наприклад, `ecdsa_private.pem`, зберігається відкрито без належних обмежень доступу, будь-який привілейований користувач або процес може прочитати його.
- 2. Компрометація генератора випадкових чисел (ГВЧ):** Стійкість ECDSA критично залежить від одноразового параметра  $k$ . Якщо зломисник може вплинути на стан ГВЧ (наприклад, підміною джерел ентропії або використанням передбачуваних seed-значень), алгоритм може згенерувати однаковий  $k$  кілька разів. Повторення  $k$  миттєво розкриває приватний ключ  $d$ .
- 3. Атаки через бічні канали:** У нативних C-реалізаціях можливі витoki інформації через час виконання, енергоспоживання чи поведінку кешу. Хоча Python (зокрема, PyCryptodome) згладжує частину цих ризиків, його скомпільовані модулі залишаються вразливими до низькорівневих операцій.
- 4. Підміна бібліотек:** Атака через LD\_PRELOAD дозволяє підвантажити власний шкідливий код або підмінити криптографічні бібліотеки, щоб скопіювати ключі та підписи ще до запуску справжніх криптографічних функцій.
- 5. Читання пам'яті процесу:** Зломисник може отримати тимчасові значення, такі як  $g$ ,  $h$  або сам  $d$ , під час підпису, якщо система дозволяє читати пам'ять процесу (наприклад, через `/proc` або механізми на зразок ptrace).

Типові сценарії атак

Сценарій атаки	Механізм недосконалості ОС	Мета атаки
Атака на "холодну" пам'ять	Відсутність повного очищення оперативної пам'яті (RAM) після вимкнення живлення.	Зчитування закритого ключа ( $d$ ) з дамів пам'яті.
Атака на свопінг	Запис чутливих даних процесу (включаючи закриті ключі) у	Вилучення ключа з нешифрованого файлу підкачки.

	файл підкачки (swapfile) на диску.	
Атака за допомогою ptrace	Дозволяє іншому процесу (з відповідними правами) перевіряти та змінювати пам'ять цільового процесу.	Зчитування закритого ключа з адресного простору процесу підписання.
Атака на /proc файлову систему	Можливість доступу до /proc/\$PID/mem для читання пам'яті процесу.	Зчитування вмісту пам'яті процесу підписання.

## Практична реалізація

Для демонстрації вразливості, пов'язаної з архітектурою ОС Linux та керуванням пам'яттю, було обрано атаку за допомогою механізму **ptrace** (з використанням GDB). Цей механізм дозволяє процесу-атакуючому приєднатися до цільового процесу (де відбувається підпис) і отримати доступ до його пам'яті.

Атака імітує ситуацію, коли зловмисник має права root або перебуває у середовищі, де політики безпеки (SELinux/AppArmor) не обмежують використання ptrace.

Код лаб3 було модифіковано таким чином, щоб після генерації ключа і перед виконанням підпису він зупинявся, утримуючи закритий ключ d у пам'яті.

```
import os
from Crypto.PublicKey import ECC
from Crypto.Signature import DSS
from Crypto.Hash import SHA256

print(f"Target PID: {os.getpid()}")
key = ECC.generate(curve='P-256')
d_value = key.d

print(f"Ключ згенеровано. {hex(d_value)} Чекаємо на команду...")
input("Натисніть Enter, щоб виконати підпис і вийти...")

message = b"Secret data to sign"
h = SHA256.new(message)
signer = DSS.new(key, 'fips-186-3')
signature = signer.sign(h)

print("Підпис виконано. Процес завершується.")
```

```
Підпис виконано? процес: 3008747507
(.venv) user@ubuntu:~/kpi/mrkm25-26/Lab4/Lab4_Sherstiuk_Vygranovsky$ python3 main.py
Target PID: 40613
Ключ згенеровано. 0xb0f2b4fb9288246a4e33a87ed5c0d7fe2db08bc10eee53469a8274bbf599c52f Чекаємо на команду...
Натисніть Enter, щоб виконати підпис i
```

Приватний ключ d: 0xb0f2b4fb9288246a4e33a87ed5c0d7fe2db08bc10eee53469a8274bbf599c52f

Приєднання GDB до цільового процесу (з іншого терміналу):

```
$ sudo gdb -p 40613
```

Перевірка регістрів процесу (якщо ключ був використаний нещодавно):

```
(gdb) info registers
```

```
(gdb) info registers
rax                0xfffffffffffffe00    -512
rbx                0x7c82498038e0      136899020732640
rcx                0x7c824971ba91      136899019782801
rdx                0x400              1024
rsi                0x115f68c0        291465408
rdi                0x0                0
rbp                0x7ffc5dae4f70      0x7ffc5dae4f70
rsp                0x7ffc5dae4f38      0x7ffc5dae4f38
r8                 0x1                1
r9                 0x0                0
r10                0x7c8249609cb8      136899018661048
r11                0x246              582
r12                0x7c8249802030      136899020726320
r13                0x7c8249801ee0      136899020725984
r14                0xba58a8          12212392
r15                0x7c82498038e0      136899020732640
rip                0x7c824971ba91      0x7c824971ba91 <__GI___libc_read+17>
eflags             0x246              [ PF ZF IF ]
cs                 0x33              51
ss                 0x2b              43
ds                 0x0                0
es                 0x0                0
fs                 0x0                0
gs                 0x0                0
fs_base            0x7c82498b2080      136899021447296
gs_base            0x0                0
```

Теоретично можна зробити:

- Пошук у купі: Якщо ми знаємо приблизний вміст ключа (наприклад, його перші байти), ми можемо шукати його. Однак, у PyCryptodome ключ `d` є об'єктом Python `int`. Нам потрібно знайти адресу об'єкта `key.d`.
- Інспекція адресного простору Python: У GDB ми можемо викликати функції Python, якщо є доступ до інтерпретатора.  
(gdb) call PyObject\_Print([address\_of\_key\_object], stdout, 0)

Зробимо дампування пам'яті для аналізу:

(gdb) generate-core-file /tmp/target\_dump.core

Шукаємо 32-байтове представлення ключа `d`:

```
tmp > target_dump.core

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded Text Data Inspector
0021D180 00 00 00 00 00 00 75 00 72 16 74 13 11 00 00 00 . . . . . u . r . t . . . . . binary 00110000
0021D190 00 00 00 00 7C 04 74 1C 11 00 00 00 00 00 00 00 . . . . . | . t . . . . . octal 060
0021D1A0 AB 02 11 00 00 00 00 72 06 56 04 64 00 7D 03 . . . . . r . v . d . } . uint8 48
0021D1B0 64 00 7D 06 53 00 74 1F 11 00 00 00 00 00 00 00 d . } . s . t . . . . . int8 48
0021D1C0 7C 01 9B 02 54 02 7C 00 6A 20 11 00 00 00 00 00 | . . . . T . | . j . . . . . uint16 30768
0021D1D0 00 00 00 00 00 00 00 00 00 00 00 00 9B 01 9D 03 . . . . . int16 30768
0021D1E0 AB 01 11 00 00 00 00 9A 06 7C 04 80 04 7C 03 . . . . . | . . . . | . . . . . uint24 6453296
0021D1F0 80 02 7C 06 82 01 7C 03 80 1C 74 07 11 00 00 00 . . | . . . | . . . . t . . . . int24 6453296
0021D200 00 00 00 00 54 03 7C 00 6A 22 11 00 00 00 00 00 . . . . . T . | . j " . . . . . uint32 811759664
0021D210 00 00 00 00 00 00 00 00 00 00 00 00 9B 01 54 04 . . . . . T . . . . . int32 811759664
0021D220 7C 04 9B 02 64 05 9D 05 AB 01 11 00 00 00 00 00 | . . . . d . . . . . uint64 3774634852169709616
0021D230 7D 03 74 13 11 00 00 00 00 00 00 00 7C 03 74 1E } . t . . . . . | . t . . . . . int64 3774634852169709616
0021D240 11 00 00 00 00 00 00 00 AB 02 11 00 00 00 00 00 . . . . . ULEB128 48
0021D250 73 07 58 06 7C 03 5F 12 11 00 00 00 00 00 00 00 s . X . | . _ . . . . . SLEB128 48
0021D260 7C 03 82 01 23 00 64 00 7D 03 64 00 7D 06 77 00 | . . . . # . d . } . d . } . w . float16 34304
0021D270 78 03 59 00 77 01 00 00 80 04 00 00 00 00 00 00 x . Y . w . . . . . bfloat16 1.4278816360970776e+34
0021D280 11 04 00 00 00 00 00 00 D0 9A D0 BB D1 8E D1 87 . . . . . float32 8.238911775038105e-10
0021D290 20 D0 B7 D0 B3 D0 B5 D0 BD D0 B5 D1 80 D0 BE D0 . . . . . float64 2.319143139305908e-56
0021D2A0 B2 D0 B0 D0 BD D0 BE 2E 20 30 78 62 30 66 32 62 . . . . . 0 x b 0 f 2 b GUID End of File
0021D2B0 34 66 62 39 32 38 38 32 34 36 61 34 65 33 33 61 4 f b 9 2 8 8 2 4 6 a 4 e 3 3 a ASCII 0
0021D2C0 38 37 65 64 35 63 30 64 37 66 65 32 64 62 30 38 8 7 e d 5 c 0 d 7 f e 2 d b 0 8 UTF-8 0
0021D2D0 62 63 31 30 65 65 65 35 33 34 36 39 61 38 32 37 b c 1 0 e e e 5 3 4 6 9 a 8 2 7 UTF-16 碎
0021D2E0 34 62 62 66 35 39 39 63 35 32 66 20 D0 A7 D0 B5 4 b b f 5 9 9 c 5 2 f . . . . . GB18030 0
0021D2F0 D0 BA D0 B0 D1 94 D0 BC D0 BE 20 D0 BD D0 B0 20 . . . . . BIG5 0
0021D300 D0 BA D0 BE D0 BC D0 B0 D0 BD D0 B4 D1 83 2E 2E . . . . . SHIFT-JIS 0
0021D310 2E 0A 5E 49 82 7C 00 00 30 73 5E 49 82 7C 00 00 . . ^ I . | . . 0 s ^ I . | . . Little Endian
0021D320 70 73 5E 49 82 7C 00 00 B0 73 5E 49 82 7C 00 00 p s ^ I . | . . s ^ I . | . .
0021D330 F0 73 5E 49 82 7C 00 00 30 74 5E 49 82 7C 00 00 . s ^ I . | . . 0 t ^ I . | . .
0021D340 70 74 5E 49 82 7C 00 00 B0 74 5E 49 82 7C 00 00 p t ^ I . | . . t ^ I . | . .
0021D350 F0 74 5E 49 82 7C 00 00 30 75 5E 49 82 7C 00 00 . t ^ I . | . . 0 u ^ I . | . .
0021D360 70 75 5E 49 82 7C 00 00 B0 75 5E 49 82 7C 00 00 p u ^ I . | . . u ^ I . | . .
0021D370 F0 75 5E 49 82 7C 00 00 50 13 5E 49 82 7C 00 00 . u ^ I . | . . P . ^ I . | . .
0021D380 30 76 5E 49 82 7C 00 00 70 76 5E 49 82 7C 00 00 0 v ^ I . | . . p v ^ I . | . .
0021D390 B0 76 5E 49 82 7C 00 00 F0 76 5E 49 82 7C 00 00 . v ^ I . | . . v ^ I . | . .
0021D3A0 30 77 5E 49 82 7C 00 00 70 77 5E 49 82 7C 00 00 0 w ^ I . | . . p w ^ I . | . .
0021D3B0 B0 77 5E 49 82 7C 00 00 F0 77 5E 49 82 7C 00 00 . w ^ I . | . . w ^ I . | . .
0021D3C0 30 78 5E 49 82 7C 00 00 70 78 5E 49 82 7C 00 00 0 x ^ I . | . . p x ^ I . | . .
0021D3D0 B0 78 5E 49 82 7C 00 00 F0 78 5E 49 82 7C 00 00 . x ^ I . | . . x ^ I . | . .
0021D3E0 30 79 5E 49 82 7C 00 00 70 79 5E 49 82 7C 00 00 0 y ^ I . | . . p y ^ I . | . .
0021D3F0 B0 79 5E 49 82 7C 00 00 F0 79 5E 49 82 7C 00 00 . y ^ I . | . . y ^ I . | . .
0021D400 10 14 5E 49 82 7C 00 00 30 7A 5E 49 82 7C 00 00 . . ^ I . | . . 0 z ^ I . | . .
```

Аналіз дампу пам'яті підтвердив наявність значення закритого ключа d у пам'яті процесу, коли він перебував у стані очікування (після генерації, але до підпису). Цей результат підтверджує низьку стійкість криптопровайдера до атак, що використовують ptrase/дампування пам'яті, коли процес, що містить закритий ключ, знаходиться у стані очікування.

Математично стійкий алгоритм не може гарантувати безпеки, якщо ОС дозволяє стороннім процесам переглядати або модифікувати її пам'ять. Таким чином, практична безпека ECDSA визначається не лише силою еліптичної криптографії, а й якістю архітектури операційної системи та правильністю побудови середовища виконання.