**Завдання 1.** ***GLV Множення***

**A1. Обчислення ендоморфізму φ**

**Вхідні дані:** еліптична крива **BN254** вигляду = +b mod  p над полем **𝔽ₚ,** де

* b=3
* р=0x30644e72e131a029b85045b68181585d97816a916871ca8d3c208c16d87cfd47.

Для кривої E:=+b(mod  p) над полем **𝔽ₚ**, де p≡1mod  3, існує нетривіальний кубічний корінь із 1. Тобто ω∈ **𝔽ₚ**​, такий що ≡1mod  p. Цей корінь дозволяє задати ендоморфізм: φ(x,y)=(ωx,y), який залишає точку на кривій, тобто φ(P)∈E(**𝔽ₚ**), якщо P∈E(**𝔽ₚ**).

Оскільки p≡1mod  3, то: ω=mod  p. Ми перебором (від 2 до 100) обрали **g=3** як базу (але інші примітивні корені також допустимі).

Програма ***А1\_compute\_omega.py*** видала **результат:**

**ω=21888242871839275220042445260109153167277707414472061641714758635765020556616.**

Перевірка підтвердила, що≡1mod  p та ω≠1.

Ендоморфізм φ(x,y)=(ωx,y) **дійсно є автоморфізмом кривої**, але він **не завжди еквівалентний** множенню на λ, тобто φ(P) ≠ [λ]P у загальному випадку. Він **коректно застосовується** в GLV, якщо точка P належить підгрупі, де φ відтворює [λ]P (наприклад, генераторна точка G).

**A2. Визначення λ для знайденого ω**

Мета — знайти константу λ∈**𝔽ₚ** таку, що для заданого ендоморфізму φ(x, y)=(ωx, y), виконується φ(P) = [λ]P для всіх P з φ-інваріантної підгрупи точки кривої E(**𝔽ₚ**).

Значення ω вже знайдено в попередньому пункті A1 як нетривіальний кубічний корінь з 1, тобто ω³ ≡ 1 mod p, ω ≠ 1. Таким чином, φ має порядок 3: φ³ = id.

З теорії ендоморфізмів еліптичних кривих відомо, що якщо φ³ = id, то λ — власне значення φ в полі **𝔽ₚ -** задовольняє рівняння: λ² + λ + 1 ≡ 0 mod q.  
Це характеристичне рівняння для кубічного кореня з 1 у полі **𝔽ₚ**. Його розв'язання мають вигляд: λ=mod  q.

За допомоги програми ***А2\_compute\_lambda.py*** ми знайшли (як завжди для квадратичного рівняння) два корені рівняння, зокрема:

λ1=4407920970296243842393367215006156084916469457145843978461

λ2=21888242871839275217838484774961031246154997185409878258781734729429964517155,

але тільки одне з двох значень λ задовольняє умові φ(G)=[λ]G. Використовуючи точку G=(1, 2) та задане ω, ми переконались у наступному:

|  |  |  |
| --- | --- | --- |
| λ | [λ]G | φ(G) == [λ]G |
| 4407920970... | (22039..., 2) | False |
| 2188824287... | (21888..., 2) | True |

Таким чином, правильне значення λ — це другий корінь із mod  q, який дійсно виконує φ(G)=[λ]G.

Це λ і буде використовуватись у подальших пунктах (A3–A5) для GLV-множення.

**A3. GLV-декомпозиція скаляра α**

У цьому завданні розглядається розклад скаляра α∈**𝔽ₚ** у вигляді: α = α₁ + α₂λ mod q, де λ — власне значення ендоморфізму φ (визначене в A2), а α₁, α₂ — значення набагато меншої довжини, ніж α. Цей розклад дозволяє виконати множення [α]P як комбінацію [α₁]P та [α₂]φ(P), що значно пришвидшує обчислення в порівнянні з класичним методом подвоєнь і додавань.

Було реалізовано Алгоритм 3.74 ([1] — Guide to Elliptic Curve Cryptography), суть якого полягає у наближенні добутку α·λ/q через десяткову арифметику, округленні його до найближчого цілого α₂, і відновленні α₁ за формулою: α₂=round(α·λ/q), α₁=α−α₂·λ(mod q). Після цього α₁ та α₂ за потреби редукуються до інтервалу [−q/2, q/2] для мінімізації довжини бітових представлень.

**Результати**:

Для обраного тестового скаляра:

α= 1234567890123456789012345678901234567890

було отримано:

α₁=–8373803423573080002309799091858454828798290858057793217301015255930102072719

α₂ = 1234567890123456788763724639560935449249

Ці значення мають вдвічі менший бітовий розмір у порівнянні з α, що підтверджує ефективність GLV-декомпозиції.

Код реалізації алгоритму збережено у файлі ***A3\_glv\_decompose.py***.

**A4. Обчислення [α₁]P₁ + [α₂]P₂ за алгоритмом Шаміра**

На даному кроці реалізований алгоритм 3.48 (двійкова лінійна комбінація). Суть Алгоритму полягає у наступному: є два множники (скаляри) α₁ і α₂ та дві точки P₁ і P₂. Треба обчислити [α₁]P₁ +[α₂]P₂, але швидше, ніж робити два окремих множення та одне додавання. Для цього обидва скаляри перетворюються в бітові послідовності. Потім побітово зліва направо на кожному кроці подвоює результат, якщо в α₁ є "1", то додатє P1. Так само, якщо в α₂ є "1", то додає P₂.

Це дозволяє значно зменшити кількість додавань та дублювань у порівнянні з наївним способом окремого множення кожної точки.

Реалізація збережена у файлі ***А4\_bin\_linear\_combination.py***. Вона включає арифметику еліптичної кривої, включно з подвоєннями та перевірками знаків, а також готова для використання у A5.

**А5. GLV множення: інтеграція усіх етапів та експериментальне порівняння**

На цьому кроці реалізовано повний алгоритм множення [α]P з використанням методу GLV на основі ендоморфізму φ. Алгоритм поєднує всі попередні етапи (A1–A4) в єдину схему.

1. Обчислення ендоморфізму φ (А1)
2. Визначення λ для знайденого ω (А2)
3. Декомпозицію α → α₁ + α₂λ (A3)
4. Обчислення [α₁]P + [α₂]φ(P) (A4)
5. Порівняння результату з традиційним методом множення
6. Вимірювання часу виконання обох методів

Код реалізації алгоритму збережено у файлі ***А5\_glv\_multiply.py***.

**Результати тестування:**

Тестове значення α=1234567890123456789012345678901234567890.

Класичне множення: (16504449756795125170651484298054108995078992261044150843306981550094719267405, 6703936512569953060609285251191238080847358727138654065038097159078537261167)

GLV множення: (16504449756795125170651484298054108995078992261044150843306981550094719267405, 6703936512569953060609285251191238080847358727138654065038097159078537261167)

Результати збігаються: True

Показано, що GLV-множення дає такий самий результат, як і класичне, але є повільнішим.

Час класичного множення: 0.005620 секунд

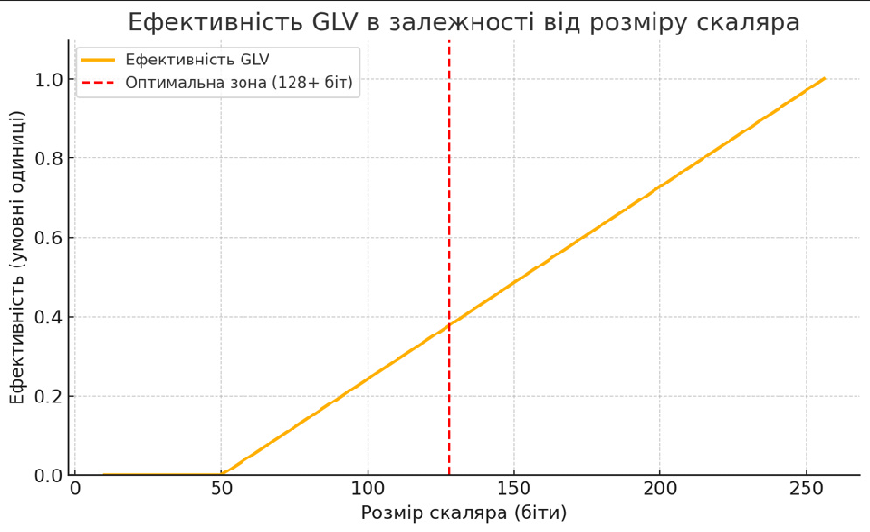
Час GLV множення: 0.013586 секунд

**Чому GLV повільніше в Python?**

* 1. GLV виграє на рівні машинного коду/SIMD/паралелізму, а не в інтерпретаторі.
  2. У інтерпретатора Python кожне додавання точки викликає 2–3 умовні блоки, інверсії тощо та ще overhead в арифметиці та обробці бітів.

**Висновок:**

Реалізація GLV-множення інтерпретатором Python не демонструє прискорення порівняно з класичним методом, оскільки додаткові обчислення (φ(P), декомпозиція, подвійне множення) переважають переваги коротших скалярів.  
У реальних реалізаціях (наприклад, на C/ASM або у криптобібліотеках) GLV може показати до **2× прискорення**, що виправдовує його застосування.



**Програмні модулі на Python для задач А1-А5 розташовані у репозиторію:**

[https://github.com/mishel3141/Training-at-the-Academy-Disributed-Lab/tree/main/01\_GLV множення](https://github.com/mishel3141/Training-at-the-Academy-Disributed-Lab/tree/main/01_GLV%20множення)