Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет прикладної математики та інформатики
Кафедра обчислювальної математики

Дипломна робота

Дослідження використання математики в технологіях криптовалют

Студент групи ПМП-42:

Солтис Сергій Петрович,

спеціальність 113-прикладна математика

Науковий керівник:

асистент

Марчук Юрій Богданович

Рецензент:

_____

Львів–2023

Зміст

Розділ 1

Вступ

Темою дипломної роботи я вибрав математику криптовалют, оскільки це цікава та актуальна тема. Наш світ все більше і більше стає цифронізованим і такі терміни як “Криптовалюта”, “Блокчейн”, “Біткоїн” входять в наш інфопростір. Для багатьох ці терміни відомі, но як це все працює всередині і яке відношення до цього має шифрування, скінченні поля, еліптичні криві та навіть прості числа? Саме це зацікавило і мене, тому спробуємо зрозуміти що це таке і чим ця тема актуальна. Також поговоримо про деякі дослідження в цій сфері і проблеми які виникають.

Математика криптовалют відноситься до математичних концепцій, алгоритмів і протоколів, які лежать в основі криптовалют і пов’язаних з ними технологій. Він охоплює різні математичні галузі, такі як криптографія, теорія ігор, розподілені системи, економічні моделі, аналіз ринку та алгоритми.

Одним з основних аспектів математики криптовалют є криптографія, яка використовує математичні алгоритми для захисту фінансових транзакцій та конфіденційності даних користувачів. Криптографічні протоколи, такі як електронний підпис, шифрування та хеш-функції, використовуються для забезпечення безпеки транзакцій та підтвердження автентичності користувачів.

Деякі з основних математичних концепцій, що лежать в основі криптовалют, включають дискретний логарифм, теорію чисел та криптографічні хеш-функції. Наприклад, алгоритм шифрування RSA (Rivest-Shamir-Adleman) базується на складності розв’язання проблеми факторизації великих цілих чисел. Ця проблема є складною для обчислення, що дозволяє забезпечити безпеку криптографічних систем.
Крім того, також використовуються алгоритми консенсусу. Це математичні протоколи, які використовуються для досягнення згоди між учасниками децентралізованої мережі, такої як блокчейн. Ці алгоритми гарантують узгодження всіх вузлів мережі щодо стану системи та дійсності транзакцій. Приклади консенсусних алгоритмів включають Proof-of-Work (PoW), Proof-of-Stake (PoS) і Practical Byzantine Fault Tolerance (PBFT), кожен зі своїми математичними принципами та властивостями.
Криптографія еліптичної кривої (ECC) — це розділ математики, який широко використовується в системах криптовалют. ECC використовує математику еліптичних кривих над кінцевими полями для забезпечення безпечного обміну ключами, цифрових підписів і шифрування. Він забезпечує ефективні та надійні криптографічні операції з меншими розмірами ключів порівняно з іншими криптографічними методами.

Багато криптовалют мають власні економічні моделі, які визначають подання, розподіл та ціну токенів. Наприклад, модель Bitcoin передбачає обмежену кількість монет (21 мільйон), алгоритмічно зменшуючу нагороду за майнінг та регулярні "половинні зменшення". Ці економічні моделі можуть використовувати математичні принципи, такі як теорія ігор, графова теорія або оптимальне управління, для моделювання розподілу, поведінки ринку та визначення ефективних стратегій.
Математика відіграє важливу роль в аналізі ринку криптовалют. Технічний аналіз використовує математичні інструменти та статистику для виявлення шаблонів цінового руху, ліній підтримки/опору та інших технічних індикаторів. Крім того, математичні моделі, такі як авторегресійні моделі, моделі Гарча та методи машинного навчання, використовуються для прогнозування цін криптовалют та ризиків інвестицій.
Звичайно ж, тема математика криптовалют, як і всі інші, стикається з рядом проблем, які активно досліджуються для знаходження вирішень. Розглянемо деякі проблеми і можливі їх вирішення з них:

Проблема 1. Безпека: Хоча криптовалюти використовують криптографічні методи для забезпечення безпеки, існує постійна загроза нових атак та проблем з безпекою. Недостатнє розуміння математичних основ криптографії або помилки при реалізації можуть призвести до вразливостей, які можуть бути використані зловмисниками.
Дослідження спрямовані на розробку нових криптографічних протоколів та покращення існуючих методів, зокрема еліптичної криптографії, гомоморфного шифрування та захисту приватності.

Проблема 2. Скейлінг: Деякі математичні алгоритми, такі як Proof-of-Work, можуть бути дуже обчислювально витратними та споживати багато енергії. Це створює проблеми зі скейлінгом та екологічним впливом криптовалют.
Дослідження розширення протоколів консенсусу. Виникає потреба у вдосконаленні протоколів консенсусу для покращення швидкості, масштабованості, енергоефективності та безпеки криптовалютних мереж. Дослідження орієнтуються на розробку нових алгоритмів консенсусу, таких як Proof-of-Stake (PoS) і Proof-of-Work (PoW) гібриди, Proof-of-Authority (PoA), а також вивчення варіацій і оптимізацій існуючих алгоритмів.

Проблема 3. Складність: Математика, що лежить в основі криптографії та алгоритмів консенсусу, може бути складною для розуміння та використання. Це може створювати перешкоди для розробників, які хочуть створити нові криптовалюти або розширити функціонал існуючих.
Однак, незважаючи на складність, **дослідження** у цій галузі приводять до розробки нових інструментів, бібліотек і платформ, які спрощують використання криптографії та алгоритмів консенсусу. Ці рішення роблять ці технології більш доступними для розробників і сприяють розвитку криптовалютного простору.

Проблема 4. Волатильність: Ринок криптовалют відомий своєю високою волатильністю, а саме різким ростом або падінням впродовж невеликого періоду часу. Волатильність напряму залежить від значень ринкової капіталізації, адже при більшій цінності, актив є менш маніпулятивним. Аналіз та передбачення цінових рухів криптовалют базуються на математичних моделях, але через низьку регуляцію, вплив новинного фону на ціну та інші фактори, ризик великих коливань залишається високим.
Дослідження та аналіз ризиків та прогнозування: Враховуючи високу волатильність

ринку криптовалют, дослідники вивчають методи аналізу ризиків та прогнозування цін. Це включає в себе використання статистичних моделей, машинного навчання та штучного інтелекту для виявлення трендів, аналізу історичних даних та передбачення цінових рухів.

Загалом, математика криптовалют забезпечує основу для безпечних транзакцій, децентралізованого консенсусу, конфіденційності та загального функціонування криптовалют і технології блокчейн. Він поєднує в собі різні математичні дисципліни, щоб забезпечити стійкість і надійність цих цифрових фінансових систем.

Розділ 2

Скінченне поле або поле Галуа

2.1 Постановка задачі

Скінченна множина — це множина, кількість елементів якої є скінченна, тобто існує натуральне число k, що є числом елементів цієї множини. В протилежному випадку множина є нескінченною. Нехай $J_n = 1, 2, ..., n$ — множина, що складається з перших n цілих чисел. Множина X називається скінченною, якщо вона еквівалентна J_n при деякому n. Число n називається кількістю елементів множини X позначається $n = |X|$.

Дві множини X та Y називаються еквівалентними, якщо існує бієктивне відображення однієї на іншу. Якщо множини еквівалентні, то цей факт записують $X \sim Y$ або $|X| = |Y|$ і кажуть, що множини мають однакові потужності.
Порожня множина є скінченною множиною, кількість елементів якої дорівнює $0 : |\varnothing| = 0$ [6]

Поле — це набір F, що складається принаймні з двох елементів із двома операціями $\oplus$ та $*$, яких задовольняються такі аксіоми:

- Множина F утворює абелеву групу (тотожність якої називається 0) відносно операції $\oplus$.
- Множина $F^* = F - \{0\} = a \in F, a \neq 0$ утворює абелеву групу (тотожність якої називається 1) під операцією $*$.
- Закон розподілу: для всіх $a, b, c \in F, (a \oplus b) * c = (a * c) \oplus (b * c)$.

Операція $\oplus$ називається додаванням (і часто позначається $+$), а операція $*$ називається множення (і часто позначається зіставленням). Як і в звичайній арифметиці, ми часто опускаємо дужки навколо добутку елементів, використовуючи конвенцію «множення перед додаванням»;
наприклад, ми інтерпретуємо $a \oplus b * c$ як $a \oplus (b * c)$.
Раціональні числа $\mathbf{Q}$, дійсні числа $\mathbf{R}$ і комплексні числа $\mathbf{C}$ — усі поля. Якщо α — ціле алгебраїчне число, тоді $\mathbf{Q}(\alpha)$ також є полем. [1]
Класифікація скінченних полів.

1. Кожне скінченне поле має p^k елементів для деякого простого числа p і деяке натуральне число k.

2. Для кожного простого числа p і натуральне число k існує скінченне поле з p^k елементів.

3. Будь-які два скінченних поля однакового розміру ізоморфні.

   Отже: існує рівно одне кінцеве поле $F(p^k)$ з p^k елементів (аж до ізоморфізму).[2]

2.2 Прості поля

Фундаментальним прикладом скінченного поля (поля Галуа) є множина F_p залишків $mod\ p$, де p є задане просте число. Тут, як і в Z_p, множина елементів $R_p = 0, 1, , p-1$, а операція $\oplus$ є додаванням $mod\ p$. Мультиплікативною операцією $*$ є множення $mod\ p$; тобто, помножити цілі числа як зазвичай, а потім взяти залишок після ділення на p.

Теорема (прості поля) Для кожного простого числа p множина $R_p = 0, 1, , p-1$ утворює поле (позначається F_p) під $mod\ p$ додавання та множення.

Теорема (єдиність простого поля) Кожне поле F з простим числом елементів p є ізоморфна F_p через відповідність $1 \oplus \cdots \oplus 1 \in F \leftrightarrow i \in F_p$.[1]

Теорема (прості підполя) Цілі числа $1, 1 \oplus 1, ...$ будь-якого скінченного поля F_q утворюють підполе $F_p \subseteq F_q$ з простим числом елементів p, де p є характеристикою F_q.
Якщо K є підполем поля L, тоді ми говоримо, що L є розширенням (або поле розширення) K. Таким чином, кожне поле є розширенням свого простого числа підполе.

Поле завжди можна розглядати як векторний простір над будь-яким його полем підполя. (Елементами поля є вектори та елементи підполя є скалярами). Якщо цей векторний простір скінченновимірний, то розмірність векторного простору називається ступенем поля над своїм підполем. Скінченне поле має бути скінченновимірним векторним простором, тому всі скінченні поля мають ступені.
Кількість елементів у скінченному полі є порядком цього поля.

Скінченне поле, оскільки воно не може містити $\mathbf{Q}$, повинно мати просте підполе форми $GF(p)$ для деякого простого p.

Теорема. Будь-яке скінченне поле з характеристикою p має p^n елементів для деякого натурального числа n.(Порядок полів p^n.)

Розглянемо поліном з коефіцієнтами в полі K, найменше розширення K, у якому поліном можна повністю розкласти на лінійні множники, це називається полем розщеплення полінома.
За приклад візьмемо поліном $x^2 + 1$, який не розкладається на множники над полем $\mathbf{R}$, але над розширенням $\mathbf{C}$ дійсних чисел розкладається, а саме $x^2 + 1 = (x + i)(x - i)$. Таким чином $\mathbf{C}$ є полем розщеплення для $x^2 + 1$.

Теорема: Якщо $f(x)$ - незвідний многочлен з коефіцієнтами у полі K, то для $f(x)$ існує поле розщеплення і будь-які два таких поля ізоморфні.

Теорема: Розщеплювальне поле $f(x) = x^{p^n} - x$, яке розглядається як поліном над полем $GF(p)$ має p^n елементів і позначається $GF(p^n)$.

Наслідок: Для кожного простого p та натурального n існує поле $GF(p^n)$ існує і є єдиним (два поля одного порядку ізоморфні). Також $GF(p^n) - \{0\} = GF(p^n)*$ є циклічною групою під множенням, а генератори цієї групи називаються примітивними елементами поля.

2.3 Побудова скінченних полів

Існує декілька способів представлення елементів скінченного поля.
Одне з представлень за допомогою многочленів. Цей спосіб ми і плануємо розглянути. Але ми наведемо інші способи представлення.

Використовуючи той факт, що поле є векторним простором над своїм простим підполем легко записати всі елементи поля у вигляді векторів $GF(4)$ є двовимірним векторним простором над $GF(2)$, тому його чотири елементи можна записати як $(0,0), (0,1), (1,0)$ і $(1,1)$.

Додавання цих елементів виконується покомпонентно (у $GF(2)$). Множення, однак, є більш складним і включає в себе дивне правило ... тому це не найкращий спосіб представлення поля.

Інша ідея, яка може бути використана як основа для представлення, полягає у тому, що ненульові елементи скінченного поля можна записати як степені примітивного елемента. Нехай ω - примітивний елемент $GF(4)$. Елементи поля $GF(4)$ є 0, ω, ω^2, ω^3. Множення легко виконується у цьому поданні (просто додаємо експоненти за модулем 3), але додавання не є очевидним. Якщо ми зможемо зв'язати ці два подання разом, ми легко зможемо виконувати як додавання, так і множення [3]

У $GF(4)$ маємо:

$$
\begin{aligned}
0 &\leftrightarrow (0,0) \\
\omega &\leftrightarrow (0,1) \\
\omega^2 = 1 + \omega &\leftrightarrow (1,1) \\
\omega^3 = 1 &\leftrightarrow (1,0)
\end{aligned}
\qquad\qquad a + b\omega \leftrightarrow (a,b)
$$

Отже ми будемо розглядати поліноми, розглянемо поле Галуа 2, 3 і 4
Скінченні поля з p^k елементами містять копію F_p. Побудова цих полів однієї змінної з

коефіцієнтами при F_p. Скінченне поле з p^k елементами буде відповідним часткою цього кільця.
Поле $GF(2)$ складається з двох елементів. Додавання та множення визначені як додавання та множення чисел по модулю 2 [4]

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

Тут все доволі просто, додавання на 0 нічого не змінює а от $1+1=2\equiv 0$ за модулем 2, множення не потребує пояснень, але можна зрозуміти що перший рядок і перший стовпчик нашої матриці будуть нулі, а другий рядок/стовпчик елементи поля

Наступним полем, яке ми розглянемо, буде поле Галуа $GF(3)$. Поле $GF(3) = 0, 1, 2$. Додавання та множення визначені як додавання та множення чисел по модулю 3. Таблиці операцій $GF(3)$ мають вигляд:

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

×	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Додавання не відрізняється і можна побачити, що це працює як зрушення елементів на n, де n – це число, яке ми додаємо(Тобто для 1 це зрушення на 1, для 2 – на 2).[4] Множення цікаве лише для елемента $2 * 2 = 4 \equiv 1$.

Наступне поле за логікою це поле $GF(4)$, але чи існує таке поле і як воно виглядає? В його існуванні можна не сумніватися, оскільки з означення скінченного поля відомо що порядок цього поля це або просте число, або степінь простого числа. Ну а $4 = 2^2$, отже спробуємо його побудувати таким же способом, як і дві попередні моделі

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

x	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Отже можна побачити, що хоч ми і зробили нашу модель, як і в двох попередніх прикладах, но це не є полем, оскільки елемент 2 не має інверсії, а в кожному полі повинні існувати нейтральний(в нашому випадку він існує - 0) і обернений елементи для не нульових елементів, $1 * 1 \equiv 1$, $3 * 3 \equiv 1$, а от 2 на що помножити немає.
В загальному для кожного поля Галуа рядки і стовпчики будуть містити лише елементи самого поля і тільки один раз. Це виконується для поля з 2 і 3 елементів, як і для всіх інших, наприклад для $GF(7)$ кожен рядок і стовпчик міститиме числа від 0 до 6 та існуватиме обернений елемент.
Отже наша модель не підходить для $GF(4)$, але це поле існує, то ж які вирішення цієї проблеми?
Тут нам допоможуть многочлени, оскільки саме з їхньою допомогою, через незвідні многочлени можна розписати елементи(розв'язки).
Многочлен $x^2 + x + 1$ незвідний до $F_2[x]$ (насправді, це єдиний незвідний квадратичний многочлен в $F_2[x]$). Розглянемо частку $F = F_2[x]/(x^2 + x + 1)$

Нехай $\alpha = x$, тоді F можна подати у вигляді $F_2[\alpha]$ де $\alpha^2 + \alpha + 1 = 0$.
Тоді елементи F дорівнюють 0, 1, α, $\alpha+1$. Поліноми в α більшого степеня можна звести до лінійних многочленів в α за допомогою співвідношення $\alpha^2+\alpha+1 = 0$. Це можна отримати з попереднього рівняння, а -1 це то саме що і 1.

Отже F має чотири елементи, і насправді не важко перевірити, що F є полем. Зокрема, $\alpha(\alpha + 1) = \alpha^2 + \alpha = \alpha + 1 + \alpha = 2\alpha + 1$, а $2\alpha \equiv 0$ а за модулем 2, тому α і $\alpha + 1$ є мультиплікативними інверсіями (і 1 є оберненим до самого себе). Це поле позначається F_4.
Зауважимо, що характеристика F_4 це 2 Це не те саме що і характеристика F_4 для цілих чисел за модулем 4: наприклад $Z/(4)$ не є полем, і $1 + 1 \neq 0$ в $Z/(4)$. [2]
Таблиці операцій у $GF(4)$ є такими:

Додавання $x + y$

x \ p	0	1	α	$1+\alpha$
0	0	1	α	$1+\alpha$
1	1	0	$1+\alpha$	α
α	α	$1+\alpha$	0	1
$1+\alpha$	$1+\alpha$	α	1	0

Множення $x \cdot y$

x \ p	0	1	α	$1+\alpha$
0	0	0	0	0
1	0	1	α	$1+\alpha$
α	0	α	$1+\alpha$	1
$1+\alpha$	0	$1+\alpha$	1	α

Ділення x / y

x \ p	1	α	$1+\alpha$
0	0	0	0
1	1	$1+\alpha$	α
α	α	1	$1+\alpha$
$1+\alpha$	$1+\alpha$	α	1

Таблиця для віднімання не надається, оскільки віднімання ідентичне додаванню, як і для кожного поля характеристики 2. У третій таблиці для ділення x на y значення x повинні бути прочитані в лівому стовпці і значення y у верхньому рядку. (Оскільки $0 \cdot z = 0$ для кожного z у кожному кільці , ділення на 0 має залишатися невизначеним).
Саме ж ділення зручно проводити за таблицею множення оскільки це будуть оберненні значення.
Зауважимо що в цій моделі у нас не виникає проблем, кожен елемент зустрічається раз і лише раз в кожному рядку і стовпчику кожної з операцій(лише 0 для множення на 0). А також в нашому полі існує нейтральний і обернений елемент.

Наступні поля Галуа це $GF(5)$ та $GF(7)$, які розписуються так само як і $GF(2)$ і $GF(3)$ від 0 до $p-1$, оскільки це прості числа, а от елементи поля $GF(8)$ і $GF(9)$ будуть іншими, оскільки це не прості числа але їхня степінь 2^3 та 3^2 і так само як і в полі $GF(4)$ потрібно розписати це за допомогою інших розв'язків, які будуть розв'язками незвідних многочленів. Наприклад поле Галуа $GF(8)$ матиме такі елементи: 0, 1, α, $\alpha+1$, α^2, α^2+1, $\alpha^2+\alpha$, $\alpha^2+\alpha+1$. Но також це можна розписати через інші елементи і ці поля просто будуть ізоморфними.
Можна зауважити, що коли поле Галуа простого числа, то елемент просто число, а коли поле це число в степені, то ми розписуємо його за допомогою поліномів.

Розділ 3

Програмна реалізація Поля Галуа

Програмну реалізацію я писав на мові програмування Python, з використаних бібліотек лише math, оскільки будемо шукати корінь з числа в одному з методів. Код програми розміщено в додатку 1

У нас є клас "Скінченного поля" і методи ініціалізації та представлення, метод представлення зручно виводить запис в виді Поле(значення елементу, модуль), а ініціалізації має перевірки на те чи значення елементу і модулю є цілочисельними значеннями, інакше виводимо помилку, також звіряємо чи це справді Поле Галуа, як ми вже вивчили це лише прості числа і прості числа в степені, а також шукаємо число по модулю, на випадок коли це число більше, а такого не може бути.

Дальше методи перевірки на просте число і на степінь простого числа, чи є простим число ми звіряємо за допомогою функції Ейлера, тобто визначаємо кількість взаємо простих чисел з нашим n. У функції phi ми використовуємо алгоритм, що базується на розкладі числа n на прості множники.

Ми починаємо зі значення result, яке початково дорівнює n. Потім ми перебираємо всі прості числа p від 2 до кореня з n. Якщо n ділиться на p, це означає, що p є одним з простих множників числа n. Ми віднімаємо від result кількість чисел, які діляться на p (тобто $result//p$), оскільки вони не є взаємно простими з n. У результаті отримаємо значення функції Ейлера.
Якщо функція Ейлера від n елементів дорівнює $n - 1$, то це число є простим і ми повертаємо "True". Дальше перевіряємо чи n є простим числом в степені. функція is_prime_power перебирає всі прості числа p від 2 до кореня з n. Для кожного простого числа p перевіряється, чи n може бути представлене як p^k для деякого натурального k. Це робиться шляхом порівняння n з p^k до тих пір, поки p^k не перевищує n. Якщо n співпадає з p^k, то це означає, що n є простим числом в степені, і функція повертає значення True. У протилежному випадку, коли n не є точним степенем жодного простого числа, функція повертає значення False.

Наступними методами будуть перевизначення наших операндів. Перший з них це порівняння двох полів, вони рівні коли значення і модуль однакові. При різних значеннях модуля Поля методи додавання, віднімання, множення, ділення і степінь зупиняють програму і виводять помилку "TypeError". В цих методах ми робимо звичайні обрахування за модулем, тому тут все просто.

Наступні методи цікавіші, це ділення і піднесення в степінь. Ділення в скінченному полі може бути складнішим, оскільки потрібно шукати обернений елемент. Тобто $a \div b$ це теж само що $a * b^{-1}$ і цим ми скористаємося. Обернений елемент в наших полях це 1, тому ми шукаємо на яке число потрібно домножити, щоб отримати 1 по модулю.

В наступному методі, піднесення до степені звіряємо чи степінь це ціле число, якщо так то у нас є такі випадки: Якщо степінь нуль, то значення 1(всі значення в 0 степені 1) Якщо степінь від'ємна, то спочатку шукаємо обернений елемент а дальше підносимо в степінь.
Піднесення в (тепер точно додатню) степінь в нас ділиться на 2 випадки, коли значення парне, то застосовується метод поділу степеня на половину. Основа підноситься до половини степеня, а потім отриманий результат підноситься до квадрату.
Наприклад, якщо основа a дорівнює 2, а power дорівнює 4, то буде обчислений результат (2 піднесене до степеня 2) піднесений до степеня 2, що дорівнює 16.
Якщо ж значення степеня непарне, то застосовується рекурсивне обчислення результату. Основа множиться на себе, підняту до степеня $(power - 1)$.
Наприклад, якщо основа a дорівнює 2, а power дорівнює 3, то буде обчислено результат основи a помноженої на (2 піднесене до степеня 2), що дорівнює 8. На цьому клас закінчується, тому розглянемо приклад з полями $GF(3, 7)$ та $GF(5, 7)$

Тут ми використовуємо всі методи і отримуємо результати:
A= FiniteField(5, 7)
False
Sum: FiniteField(1, 7)
Diff: FiniteField(2, 7)
Mult: FiniteField(1, 7)
Div: FiniteField(4, 7)
A to power -3: FiniteField(6, 7)
B to power 4 FiniteField(4, 7)

Розділ 4

Еліптичні криві

Еліптичні криві (Elliptic curves) є важливим об'єктом в алгебрі та математиці, а також в криптографії. Вони виникають з рівняння, яке має форму:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

де a_i - константи, які визначають конкретну криву. Це рівняння визначає множину точок (x, y), які задовольняють йому. [5]
Одна з ключових властивостей еліптичних кривих полягає в тому, що вони утворюють абелеву групу з операцією додавання точок. Додавання точок на еліптичних кривих визначається геометрично і має свої особливості. Воно дозволяє обчислювати суму точок і знаходити добуток точки на ціле число. Еліптичні криві також мають нейтральний елемент, який називається "безкінечністю"і позначається символом O. Цей елемент виступає як ідентичний елемент для операції додавання. Коли додається точка до O, результатом є сама точка.

У криптографії еліптичні криві знаходять широке застосування, зокрема в алгоритмах еліптичного криптосистеми (Elliptic Curve Cryptography, ECC). ECC заснована на складності обчислення дискретного логарифма на еліптичних кривих, що робить її ефективною та безпечною для використання в криптографічних протоколах, таких як шифрування, цифровий підпис та обмін ключами.

Ступінь захисту (на кожен біт ключа)	*Мінімальна довжина ключа (в бітах)*	
	RSA/DSA/DH	**ECC**
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

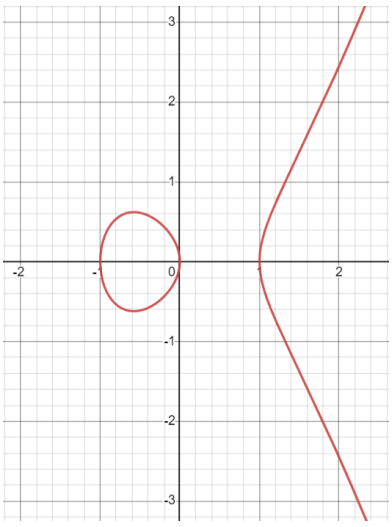
Основною перевагою криптосистем на еліптичних кривих у порівнянні із звичайними асиметричними алгоритмами є те, шо вони забезпечують еквівалентний захист за меншої довжини ключа, як це показано вище Розглянемо рівняння еліптичної кривої у спрощеному вигляді (рівняння Вейєрштрасса):

$$y^2 = x^3 + ax + b$$

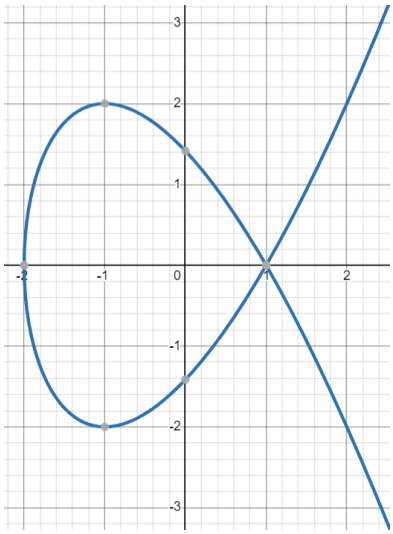
Залежно від значень параметрів a і b еліптичні криві можуть приймати на площині різні форми. Так як $y = \pm\sqrt{x^3 + ax + b}$, то графік кривої симетричний відносно Ox.

Дискримінант рівняння: $D = (\frac{a}{3})^3 + (\frac{b}{2})^2$.

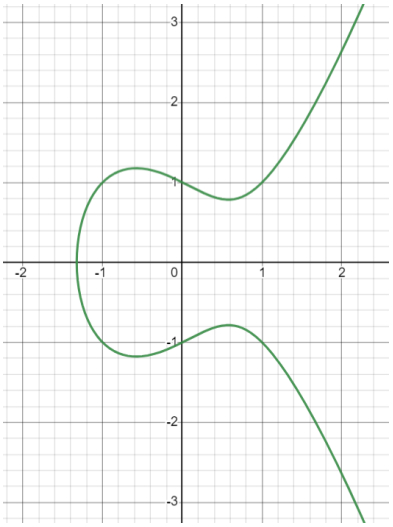
- $D < 0$ – три різних дійсних корені (графік 1);
- $D = 0$ – три дійсних корені, два з яких однакові (графік 2 – сингулярна крива, такі криві виключають з розгляду);
- $D > 0$ – один дійсний корінь та два комплексних (графік 3) [6]



$y^2 = x^3 - x$(Графік 1)



$y^2 = x^3 - 3x + 2$(Графік 2)



$y^2 = x^3 - x + 1$(Графік 3)

У реальних криптосистемах використовуються еліптичні криві над скінченним полем p, що описуються рівнянням:

$$y^2 \equiv x^3 + ax + b \ (mod\ p),$$

де (x, y) – точки еліптичної кривої, a, b – параметри кривої, p – просте число ($p \neq 2,\ p \neq 3$). При цьому параметри кривої a та b мають задовольняти умову:

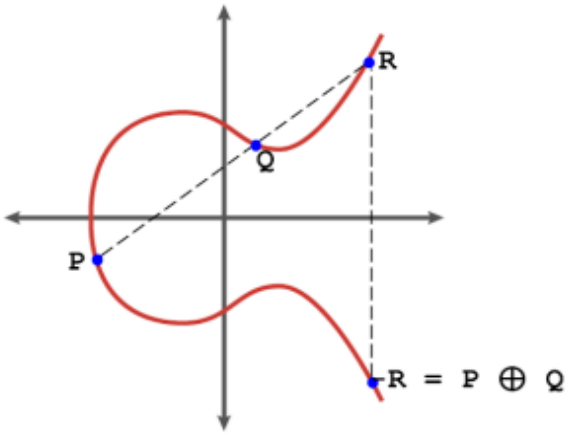
$$4a^3 + 27b^2 \neq 0 \ (mod\ p).$$

Позначимо через $E_p(a,\ b)$ множину точок еліптичної кривої. У множину точок еліптичної кривої також включається нескінченно віддалена точка O. Точка належить еліптичній кривій, якщо пара чисел $(x,\ y)$ задовільняє рівняння.
Кількість точок кривої називається порядком кривої. [6]

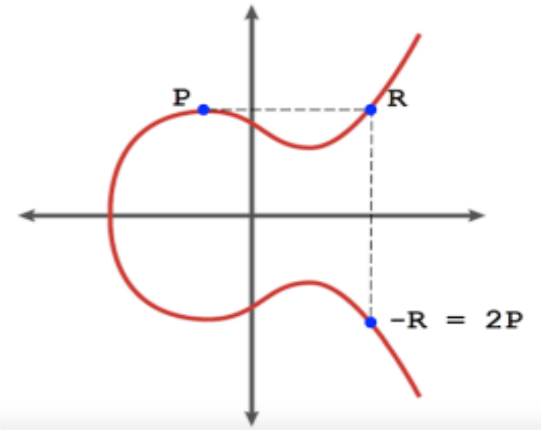
4.1 Додавання двох точок кривої

Нехай маємо дві точки P і Q на еліптичній криві. Щоб отримати нову точку R, яка є сумою точок P і Q, застосовуються такі кроки:

1. Якщо P і Q є різними точками, проводиться пряма лінія, яка проходить через ці точки. Ця пряма лінія перетинає криву на третій точці R. Проведемо через точку R вертикальну пряму до перетину з 4 кривою у точці $-R = P + Q$. Отже, сумою двох точок P та Q буде точка, обернена до третьої точки перетину еліптичної кривої і прямої, що проходить через задані точки.



2. Якщо P і Q є однаковими точками (тобто $P = Q$), проводиться пряма лінія, яка проходить через цю точку. Ця пряма лінія має дотикатися кривої в точці P, і точка R - це точка перетину цієї дотичної з кривою. При $P = Q$ січна перетворюється на дотичну, тому точка $2P$ є оберненою до точки R. [6]



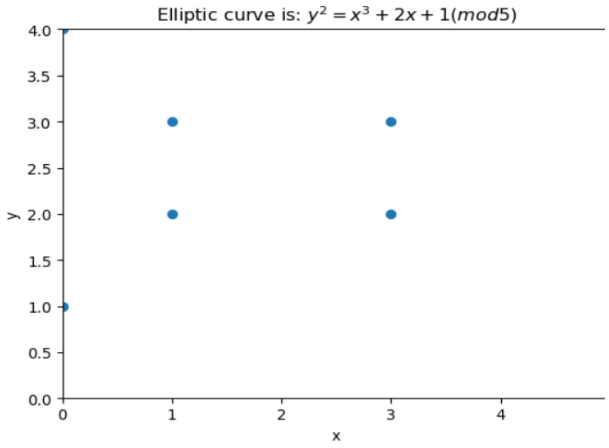
Координати $-R(x_3, y_3)$ визначаються за формулами, де λ – кутовий коефіцієнт січної, що проведена через точки $P(x_1, y_1)$ та $Q(x_2, y_2)$.

Додавання точок (якщо P ≠ Q)	*Подвоєння точки (якщо P = Q)*
$x_3 = \lambda^2 - x_1 - x_2 (mod\ p);$	$x_3 = \lambda^2 - 2x_1\ (mod\ p);$
$y_3 = \lambda(x_1 - x_3) - y_1 (mod\ p);$	$y_3 = \lambda(x_1 - x_3) - y_1 (mod\ p);$
$\lambda = \frac{y_2 - y_1}{x_2 - x_1} (mod\ p).$	$\lambda = \frac{3x_1^2 + a}{2y_1} (mod\ p).$

3. Якщо P і Q мають координати, що не належать кривій, або $P = O$ (бескінечність), тоді R вважається рівним Q (або P, якщо $P = O$).

Отримана точка R є результатом операції додавання точок P і Q на еліптичній криві. Важливо враховувати, що координати точок можуть бути елементами поля, такого як раціональні числа або скінченні поля, і операція додавання виконується згідно з правилами поля.
Операція додавання точок на еліптичних кривих має комутативність (тобто $P+Q = Q+P$) та асоціативність (тобто $(P + Q) + R = P + (Q + R)$).

Приклад: Множина точок $E_5(2, 1)$ еліптичної кривої $y^2 \equiv x^3 + 2x + 1(mod\ 5)$ складається з 6 точок. Порядок кривої – 7.



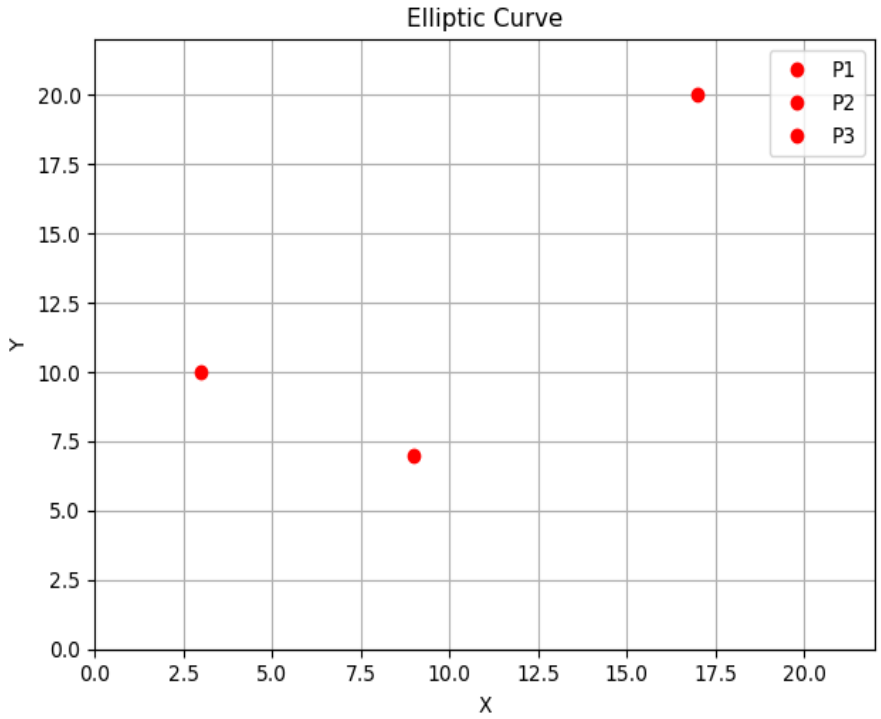
Як можна побачити на рисунку зображено усі точки, що задовольняють умови кривої, це точки $(0, 1), (0, 4), (1, 2), (1, 3), (3, 2), (3, 3)$. Перевірити це ж можна простим підставленням
$1^2 \equiv 0^3 + 2 * 0 + 1$ або ж $3^2 \equiv 3^3 + 2 * 3 + 1 \to 9 \equiv 27 + 6 + 1(mod\ 5) \to = 4 \equiv 4$

Тепер розглянемо ж приклад з додаванням, спочатку обрахуємо це математично самі, оскільки ми візьмемо не великі значення, а дальше подивимося як це працює в програмі і чи наші відповіді співпадають.

Математично

Візьмемо ось таке рівняння для еліптичної кривої $y^2 \equiv x^3 + x + 1(mod\ 23)$ і перевіримо чи точки $P(3, 10)$ та $Q(9, 7)$ належать кривій і якщо належать знайдемо їхню суму.
Підставимо значення у рівняння еліптичної кривої і удостовіримось що точки належать кривій
$10^2 \equiv 3^3 + 3 + 1(mod\ 23) \to 100 mod\ 23 \equiv 31(mod\ 23) \to 8 \equiv 8$; $7^2 \equiv 9^3 + 9 + 1(mod\ 23) \to 49 mod\ 23 \equiv 739(mod\ 23) \to 3 \equiv 3$.
Перевірили, виконаємо додавання – знаходимо кутовий коефіцієнт січної

$\lambda = \frac{y_2-y_1}{x_2-x_1}(mod\ p) = \frac{7-10}{9-3}(mod\ 23) = -3/6(mod\ 23) = -1/2(mod\ 23) = 22/2(mod 23) = 11.$
Тепер знаходимо точки x та y:
$x_3 = \lambda^2 - x_1 - x_2(mod\ p) = 121 - 3 - 9(mod\ 23) = 109(mod\ 23) = 17;$
$y_3 = \lambda(x_1 - x_3) - y_1(modp) = 11(3?17)?10(mod 23) = -164(mod 23) = 20.$
Отже $P + Q = (3, 10) + (9, 7) = (17, 20).$
Код до цієї програми розміщено в додатку під номером 2.
І вивід, який ми отримаємо:
$p1 + p2 = (17, 20)$
$p1 == p2 : False$
$p1! = p2 : True$
Також малюнок матиме такий вигляд:



Код працює за такою логікою:

1. Клас EllipticCurvePoint визначає точку на еліптичній кривій. Конструктор `__init__` ініціалізує координати точки (x і y), параметри еліптичної кривої (a і b), які виступають коефіцієнтом біля x та вільним членом та поле простого числа (p).

2. Метод `__eq__` перевіряє, чи дві точки належать одній еліптичній кривій, порівнюючи координати x, y, a, b і p точок.

3. Метод `__ne__` перевіряє чи точки не належать кривій, використовуючи метод `__eq__`

4. Метод `__add__` виконує додавання двох точок на еліптичній кривій. Він перевіряє, чи точки належать одній еліптичній кривій, обчислює коефіцієнт наклону прямої або дотичної, залежно від того, чи точки рівні або різні, і обчислює координати третьої точки додавання.

5. Метод `__str__` повертає рядок, що представляє точку у форматі (x, y).

6. Задається поле простого числа p (у випадку даного прикладу використовується значення 23).

7. Створюються дві точки $p1$ і $p2$ з відповідними координатами і параметрами еліптичної кривої.

8. Виконується операція додавання двох точок: $p3 = p1 + p2$.

9. Малюємо графік з точками p_1, p_2, p_3.

Розділ 5

Криптографія з допомогою еліптичних кривих

ECDSA (Elliptic Curve Digital Signature Algorithm) - це криптографічно безпечна схема цифрового підпису, заснована на криптографії еліптичних кривих (ECC). ECDSA спирається на математику циклічних груп еліптичних кривих над скінченними полями та на складність задачі ECDLP (задача дискретного логарифмування еліптичних кривих). Алгоритм підпису/перевірки ECDSA ґрунтується на точковому множенні EC і працює, як описано нижче. Ключі і підписи ECDSA коротші, ніж в RSA для того ж рівня безпеки.[7]

ECDSA використовує криптографічні еліптичні криві (ЕК) над скінченними полями в класичній формі Вейєрштрасса. Ці криві описуються параметрами домену ЕК, визначеними різними криптографічними стандартами, такими як SECG: SEC 2 і Brainpool.
Параметри алгоритму

1. Вибір хеш-функції $H(x)$. Для використання алгоритму необхідно, щоб повідомлення, яке підписується, було числом. Хеш-функція повинна перетворити будь-яке повідомлення в послідовність бітів, які можна потім перетворити в число.

2. Вибір великого простого числа n - порядок однієї з циклічних підгруп групи точок еліптичної кривої. Зауваження: Якщо розмірність цього числа в бітах менше розмірності в бітах значень хеш-функції $H(x)$ то використовуються тільки ліві біти значення хеш-функції.

3. Простим числом p позначається характеристика поля координат F_p. [8]

Генерування ключів ECDSA Нехай E - еліптична крива, визначена над F_p, і P - точка простого порядку n кривої $E(F_p)$. Крива E і точка P є системними параметрами. Число p - просте. Кожен користувач - умовно назвемо його Аліса - конструює свій ключ за допомогою наступних дій:

1. Вибирає випадкове або псевдовипадкове ціле число x з інтервалу $[1,\ n-1]$.

2. Обчислює (кратне) $Q = xP$.

Відкритим ключем користувача Аліси A є точка Q , а закритим - x

Створення підпису ECDSA

У цьому розділі описано кроки створення та перевірки підписів за допомогою ECDSA.

Сутність A з атрибутами домену $D = (q, FR, a, b, G, n, h)$ та відповідною парою ключів (x, Q) виконує наступні дії для підписання повідомлення m.

1. Вибирається випадкове або псевдовипадкове ціле число k, де k лежить у діапазоні від 1 до $n - 1$.
2. Перетворити x_1 у ціле число, обчисливши $kG = (x1, y1)$.
3. Обчислити $r = x_1 \, mod \, n$ і якщо $r = 0$, то перейти до кроку 1.
4. Обчислити $k - 1 \, mod \, n$.
5. Обчислити $s = k - 1(h + xr) \, mod \, n$. Якщо s дорівнює 0, то перейти до кроку 1.
6. Повідомлення m містить підпис $A(r, s)$.

Перевірка підпису ECDSA

B отримує автентичну копію параметрів домену $A = (q, FR, a, b, G, n, h)$ і відповідний відкритий ключ Q, щоб перевірити підпис $A(r, s)$ на m. Рекомендується, щоб B також перевірив X і Q. Далі B виконує наступні дії:

1. Перевіряє, чи цілі числа r та s знаходяться у діапазоні $[1, n - 1]$.
2. Обчислити значення хеш-функції h від повідомлення.
3. Обчислити $w = s - 1 \, mod \, n$.
4. Обчислити $u1 = hw \, mod \, n$ та $u2 = rw \, mod \, n$.
5. Обчислити $X = u_1 \cdot G + u_2 \cdot Q$.
6. Якщо $X = O$, то відкинути підпис. Якщо ні, то обчислити $v = x1 \, mod \, n$ і перетворити координату Xx_1 у ціле число.
7. Прийняти сигнатуру тоді і тільки тоді, коли $v = r$.[9]

Справді можна побачити, що даний алгоритм дійсно успішно засвідчує цифровий підпис. Якщо підпис (rs), повідомлення m було дійсно згенеровано з використанням секретного ключа, то $s \equiv k^{-1}(h + xr)(mod \; q)$.
Розкриваючи дужки отримаємо:

$$k \equiv s^{-1}(h + xr) \equiv s^{-1}h + s^{-1}rx = u_1 + u_2x(mod \; q).$$

Тоді $u_1 \cdot P + u_2 \cdot Q = u_1 + u_2x \cdot P = kP$ і значить $v = r$, що і вимагається для підтвердження підпису.

Переваги ECDSA це високий рівень безпеки, висока продуктивність програми, висока швидкість перевірки

Також я вирішив використати бібліотеку ecdsa взяту з Github для перевірки різних кривих і їхньої швидкості. Оскільки вже є готова бібліотека, я не писав нічого нового, лише запустив її і розгляну результати [10]

Ця бібліотека забезпечує генерацію ключів, підписання, перевірку та спільний секрет для п'яти популярних кривих NIST "Suite B"$GF(p)$ (просте поле) з довжиною ключів 192, 224, 256, 384 і 521 біт. «Короткі назви» цих кривих, prime192v1, secp224r1, prime256v1,

secp384r1i secp521r1. Він включає 256-бітну криву, secp256k1 яку використовує біткойн. Також є підтримка звичайних (нескручених) варіантів кривих Брейнпула від 160 до 512 біт. "Короткі назви"цих кривих: brainpoolP160r1, brainpoolP192r1, brainpoolP224r1, brainpoolP256r1, brainpoolP320r1, brainpoolP384r1, brainpoolP512r1. Також включено кілька малих кривих стандарту SEC, а саме: secp112r1, secp112r2,secp128r1, і secp160r1. Для кривих Ed25519 і Ed448 також підтримується генерація ключів, підтвердження та перевірка.

У наведеній нижче таблиці показано, скільки часу потрібно цій бібліотеці для генерації пар ключів (keygen), підписання даних (sign), перевірки цих підписів (verify), отримання спільного секрету (ecdh) і перевірки підписів без попереднього обчислення ключа (no PC verify) . . Усі ці значення вказані в секундах.

Для зручності також наведено обернені значення цих значень: скільки ключів за секунду можна згенерувати (keygen/s), скільки підписів можна зробити за секунду (sign/s), скільки підписів можна перевірити за секунду (verify/s), скільки спільних секретів можна бути отримані за секунду (ecdh/s), і скільки підписів без попереднього обчислення ключа можна перевірити за секунду (no PC verify/s). Розмір необробленого підпису (зазвичай найменший спосіб кодування підпису) також надається в стовпці siglen.

Використовши цю біблотеку на власному комп'ютері. На Intel(R) Core(TM) i3-6006U CPU @ 2.00GHz ГГц я отримую таку продуктивність:

	siglen	keygen	keygen/s	sign	sign/s	verify	verify/s	no PC verify	no PC verify/s
NIST192p:	48	0.00148s	674.64	0.00334s	299.35	0.00423s	236.20	0.00671s	149.11
NIST224p:	56	0.00202s	494.53	0.00256s	389.99	0.00420s	237.85	0.00730s	136.93
NIST256p:	64	0.00235s	426.22	0.00285s	350.86	0.00443s	225.50	0.00921s	108.58
NIST384p:	96	0.02078s	48.13	0.00444s	224.98	0.00949s	105.40	0.01675s	59.71
NIST521p:	132	0.02913s	34.33	0.00791s	126.34	0.01472s	67.95	0.03085s	32.42
SECP256k1:	64	0.00211s	473.20	0.00223s	448.95	0.00462s	216.45	0.00842s	118.76
BRAINPOOLP160r1:	40	0.00106s	943.24	0.00114s	875.64	0.00196s	511.08	0.00493s	202.87
BRAINPOOLP192r1:	48	0.00140s	715.05	0.00159s	629.24	0.00255s	392.66	0.00508s	196.71
BRAINPOOLP224r1:	56	0.00167s	598.33	0.00210s	477.18	0.00314s	318.94	0.00642s	155.85
BRAINPOOLP256r1:	64	0.00226s	442.25	0.00230s	433.88	0.00336s	297.18	0.00927s	107.83
BRAINPOOLP320r1:	80	0.00264s	379.15	0.00290s	344.96	0.00534s	187.29	0.01304s	76.71
BRAINPOOLP384r1:	96	0.00402s	248.74	0.00469s	213.29	0.00904s	110.63	0.01759s	56.85
BRAINPOOLP512r1:	128	0.02903s	34.45	0.00824s	121.42	0.01494s	66.91	0.03292s	30.37
SECP112r1:	28	0.00058s	1737.15	0.00078s	1290.19	0.00120s	830.54	0.00268s	373.60
SECP112r2:	28	0.00058s	1711.95	0.00062s	1620.01	0.00120s	836.20	0.00242s	412.54
SECP128r1:	32	0.00078s	1289.90	0.00093s	1075.00	0.00231s	433.03	0.00307s	325.70
SECP160r1:	42	0.00110s	910.08	0.00112s	895.80	0.00212s	471.03	0.00428s	233.40
Ed25519:	64	0.00303s	330.56	0.00176s	568.71	0.00462s	216.41	0.01271s	78.69
Ed448:	114	0.03024s	33.07	0.01341s	74.60	0.01535s	65.16	0.03916s	25.54
BRAINPOOLP160t1:	40	0.00106s	940.72	0.00126s	794.71	0.00192s	519.78	0.00434s	230.25
BRAINPOOLP192t1:	48	0.00140s	713.81	0.00148s	675.66	0.00250s	399.25	0.00576s	173.66
BRAINPOOLP224t1:	56	0.00163s	612.25	0.00225s	443.86	0.00348s	286.99	0.00724s	138.04
BRAINPOOLP256t1:	64	0.00208s	480.25	0.00300s	333.63	0.00503s	198.80	0.01016s	98.43
BRAINPOOLP320t1:	80	0.00261s	383.68	0.00301s	331.93	0.00627s	159.43	0.01258s	79.50
BRAINPOOLP384t1:	96	0.00436s	229.58	0.00472s	211.73	0.00861s	116.16	0.01787s	55.96
BRAINPOOLP512t1:	128	0.03552s	28.15	0.00831s	120.37	0.01477s	67.69	0.03585s	27.90

```
                    ecdh     ecdh/s
       NIST192p:   0.00441s    226.59
       NIST224p:   0.00702s    142.46
       NIST256p:   0.00865s    115.65
       NIST384p:   0.01449s     69.00
       NIST521p:   0.02791s     35.83
     SECP256k1:    0.00752s    132.95
BRAINPOOLP160r1:   0.00363s    275.70
BRAINPOOLP192r1:   0.00500s    200.07
BRAINPOOLP224r1:   0.00532s    188.11
BRAINPOOLP256r1:   0.00857s    116.67
BRAINPOOLP320r1:   0.01051s     95.12
BRAINPOOLP384r1:   0.01583s     63.16
BRAINPOOLP512r1:   0.02598s     38.49
      SECP112r1:   0.00230s    434.88
      SECP112r2:   0.00238s    419.68
      SECP128r1:   0.00252s    396.83
      SECP160r1:   0.00424s    235.86
BRAINPOOLP160t1:   0.00505s    197.83
BRAINPOOLP192t1:   0.00446s    224.06
BRAINPOOLP224t1:   0.00586s    170.73
BRAINPOOLP256t1:   0.00800s    125.04
BRAINPOOLP320t1:   0.01096s     91.23
BRAINPOOLP384t1:   0.01476s     67.76
BRAINPOOLP512t1:   0.03324s     30.09
```

Проведемо порівняння алгоритмів ECDSA на різних еліптичних кривих за даними, які надані.

1. Підписування (sign):
   Найшвидші алгоритми: SECP112r1, SECP112r2, SECP128r1, BRAINPOOLP160r1, BRAINPOOLP160t1.
   Найповільніші алгоритми: NIST384p, BRAINPOOLP384r1, Ed448.

2. Перевірка сигнатури (verify):
   Найшвидші алгоритми: SECP112r1, SECP112r2, SECP128r1, BRAINPOOLP160r1, BRAINPOOLP160t1.
   Найповільніші алгоритми: NIST384p, BRAINPOOLP384r1, Ed448.

3. Генерація ключів (keygen):
   Найшвидші алгоритми: SECP112r1, SECP112r2, SECP128r1, BRAINPOOLP160r1, BRAINPOOLP160t1.
   Найповільніші алгоритми: NIST384p, BRAINPOOLP384r1, Ed448.

4. Перевірка сигнатури без контролю точки (no PC verify):
   Найшвидші алгоритми: SECP112r1, SECP112r2, SECP128r1, BRAINPOOLP160r1, BRAINPOOLP160t1.
   Найповільніші алгоритми: NIST384p, BRAINPOOLP384r1, Ed448.

5. Для порівняння швидкості "ecdh"на різних кривих, ми розглянемо час виконання на кожній кривій.

   Еліптичний Diffie-Hellman (ecdh):
   Найшвидші алгоритми: SECP112r1, SECP112r2, SECP128r1, BRAINPOOLP160r1, BRAINPOOLP160t1.
   Найповільніші алгоритми: BRAINPOOLP512t1, Ed448.

Загалом, з вищевказаної аналітики можна зробити такі висновки:

Для виконання операцій підписування, перевірки сигнатури, генерації ключів та перевірки сигнатури без контролю точки на еліптичних кривих, найшвидшими алгоритмами є SECP112r1, SECP112r2, SECP128r1, BRAINPOOLP160r1 та BRAINPOOLP160t1. Найповільнішими алгоритмами є NIST384p, BRAINPOOLP384r1 та Ed448.

Щодо ефективності "ecdh"(еліптичного Diffie-Hellman), найшвидшими алгоритмами є SECP112r1, SECP112r2, SECP128r1, BRAINPOOLP160r1 та BRAINPOOLP160t1.
Найповільнішими алгоритмами є BRAINPOOLP512t1 та Ed448.

Кращі алгоритми:

Для швидкості підписування (signing) та перевірки підпису (verification) можна виокремити алгоритми Ed25519 та Ed448, які мають високу продуктивність і низьку витрату ресурсів.

З погляду безпеки, алгоритми, які використовують більш довгі криві, такі як NIST521p, можуть мати більшу стійкість до атак на базі обчислювальних ресурсів, таких як квантові обчислювання.

Але варто враховувати не тільки швидкість виконання, але й безпеку, підтримку алгоритмів сторонніми системами та інші вимоги конкретного застосування

Розділ 6

Висновок

Дослідження використання математики в технологіях криптовалют, зокрема в контексті поля Галуа, еліптичних кривих (ECC) та алгоритму ECDSA, підтверджують важливість математичних концепцій і методів для забезпечення безпеки та приватності в криптографічних системах.

Дослідження поля Галуа, що є скінченним полем, дозволяють побудовувати криптографічні алгоритми, зокрема алгоритми шифрування та підписування, які базуються на арифметиці в цих полях. Використання полів Галуа дозволяє забезпечити високий рівень захисту від атак, основаних на математичних алгоритмах, таких як факторизація чисел.

Еліптичні криві (ECC) є іншим важливим математичним інструментом, який використовується в криптографії криптовалют. Вони в основному використовуються для побудови алгоритмів підписування, як от ECDSA (еліптична крива цифрового підпису). ECC забезпечує високий рівень безпеки при використанні коротших ключів порівняно з іншими криптографічними системами, що робить його особливо ефективним для обміну інформацією в області криптовалют.

Алгоритм ECDSA (еліптична крива цифрового підпису) є одним з найпоширеніших алгоритмів підписування, використовуваних в технологіях криптовалют. Він базується на властивостях еліптичних кривих і забезпечує надійний і безпечний механізм підписування та перевірки підпису. ECDSA гарантує конфіденційність, цілісність та автентичність транзакцій, що є важливими функціями в криптовалютних системах.

Отже, дослідження використання математики у технологіях криптовалют підтверджують їх вагому роль у забезпеченні безпеки, приватності та надійності. Ці математичні концепції дозволяють побудовувати криптографічні системи, які забезпечують захист від атак і забезпечують безпечний обмін інформацією у сфері криптовалют.

Список літератури

[1] *Скінченна множина* [Електронний ресурс]. – дата візиту 01.06.2023. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Скінченна_множина

[2] *Fields* [Електронний ресурс]. – дата візиту 03.06.2023. – Режим доступу до ресурсу: MIT6_451S05_FullLecNotes.pdf

[3] *Classification of Finite Fields* [Електронний ресурс]. – дата візиту 04.06.2023. – Режим доступу до ресурсу: Finite Fields

[4] *FinFields* [Електронний ресурс]. – дата візиту 048.06.2023. – Режим доступу до ресурсу: FinFields

[5] *Поле Галуа* [Електронний ресурс]. – дата візиту 05.06.2023. – Режим доступу до ресурсу: Поле Галуа

[6] *Еліптична крива* [Електронний ресурс]. – дата візиту 06.06.2023. – Режим доступу до ресурсу: Еліптичні криві

[7] *Криптографія на еліптичних кривих* [Електронний ресурс]. – дата візиту 07.06.2023. – Режим доступу до ресурсу: КРИПТОГРАФІЧНІ ПЕРЕТВОРЕННЯ В ГРУПАХ ТОЧОК ЕЛІПТИЧНИХ КРИВИХ

[8] *ECDSA: Elliptic Curve Signatures* [Електронний ресурс]. – дата візиту 08.06.2023. – Режим доступу до ресурсу: ECDSA

[9] *Elliptic Curve Digital Signature Algorithm* [Електронний ресурс]. – дата візиту 09.06.2023. – Режим доступу до ресурсу: Signature generation algorithm

[10] *Elliptic Curve Signature* [Електронний ресурс]. – дата візиту 10.06.2023. – Режим доступу до ресурсу: Signature generation/verification algorithm

[11] *Python ECDSA* [Електронний ресурс]. – дата візиту 11.06.2023. – Режим доступу до ресурсу: Python ECDSA

Посилання на репозиторій GitHub у який завантажено увесь код дослідження https://github.com/kpm-lnu/student-applications/tree/develop/2022-2023/PMP-42/coursework/Serhii%20Soltys

Розділ 7

Додаток

Додаток 1

```
import math

class FiniteField:
    def __init__(self, value, modulus):
        if not isinstance(value, int) or not isinstance(modulus, int):
            raise TypeError("Both value and modulus must be integers")
        if not self._is_prime(modulus) and not self._is_prime_power(modulus):
            raise ValueError("Modulus must be a prime or prime power")
        self.value = value % modulus
        self.modulus = modulus

    def __repr__(self):
        return f"FiniteField({self.value}, {self.modulus})"

    def _phi(self, n):
        result = n
        p = 2
        while p * p <= n:
            if n % p == 0:
                while n % p == 0:
                    n = n // p
                result -= result // p
            p += 1
        if n > 1:
            result -= result // n
        return result
```

```
def _is_prime(self, n):
    if n <= 1:
        return False
    return self._phi(n) == n - 1

def _is_prime_power(self, n):
    if n <= 1:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            while n % i == 0:
                n = n // i
            if n != 1:
                return False
            return self._is_prime(i)
    return self._is_prime(n)

def __eq__(self, other):
    if isinstance(other, FiniteField):
        return self.value == other.value and self.modulus == other.modulus
    return False

def __add__(self, other):
    if isinstance(other, FiniteField) and self.modulus == other.modulus:
        sum = (self.value + other.value) % self.modulus
        return FiniteField(sum, self.modulus)
    raise TypeError("Both operands must be instances of FiniteField class with the same

def __sub__(self, other):
    if isinstance(other, FiniteField) and self.modulus == other.modulus:
        difference = (self.value - other.value) % self.modulus
        return FiniteField(difference, self.modulus)
    raise TypeError("Both operands must be instances of FiniteField class with the same

def __mul__(self, other):
    if isinstance(other, FiniteField) and self.modulus == other.modulus:
        product = (self.value * other.value) % self.modulus
        return FiniteField(product, self.modulus)
    raise TypeError("Both operands must be instances of FiniteField class with the same
```

```
    def __truediv__(self, other):
        if isinstance(other, FiniteField) and self.modulus == other.modulus:
            reciprocal_value = self._find_reciprocal_element(other.value)
            quotient = (self.value * reciprocal_value) % self.modulus
            return FiniteField(quotient, self.modulus)
        raise TypeError("Both operands must be instances of FiniteField class with the same

    def __pow__(self, power):
        if isinstance(power, int):
            if power == 0:
                return FiniteField(1, self.modulus)
            elif power < 0:
                reciprocal_value = self._find_reciprocal_element(self.value)
                return FiniteField(reciprocal_value, self.modulus) ** abs(power)
            elif power % 2 == 0:
                half = self ** (power // 2)
                return half * half
            else:
                return self * (self ** (power - 1))
        raise TypeError("The exponent must be an integer")

    def _find_reciprocal_element(self, value):
        for i in range(1, self.modulus):
            if (value * i) % self.modulus == 1:
                return i
        raise ValueError(
            f"Cannot find reciprocal element for the value {value} in the finite field with


a = FiniteField(5, 7)
b = FiniteField(3, 7)

print("A=", a)
print(a == b)

sum = a + b
print("Sum:", sum)

difference = a - b
```

```
print("Diff:", difference)

product = a * b
print("Mult:", product)

quotient = a / b
print("Div:", quotient)

power = a ** -3
print("A to power -3:", power)

power1 = b ** 4
print("B to power 4", power1)
```

Додаток 2

```
import matplotlib.pyplot as plt
import numpy as np


class EllipticCurvePoint:
    def __init__(self, x, y, a, b, p):
        self.x = x
        self.y = y
        self.a = a
        self.b = b
        self.p = p

    def __eq__(self, other):
        if isinstance(other, EllipticCurvePoint):
            return (
                    self.x == other.x and
                    self.y == other.y and
                    self.a == other.a and
                    self.b == other.b and
                    self.p == other.p
            )
        return False

    def __ne__(self, other):
```

```
        return not self.__eq__(other)

    def __add__(self, other):
        if isinstance(other, EllipticCurvePoint):
            # Перевірка, чи точки належать одній еліптичній кривій
            if self.a != other.a or self.b != other.b or self.p != other.p:
                raise ValueError("Точки не належать одній еліптичній кривій")

            # Додавання точок на еліптичній кривій
            if self == other:
                # Перевірка, чи точка є нейтральним елементом
                if self.y == 0:
                    return EllipticCurvePoint(float('inf'), float('inf'), self.a, self.b, se

                # Обчислення коефіцієнта наклона дотичної
                slope = ((3 * self.x ** 2 + 2 * self.a * self.x + self.b) * pow(2 * self.y,
            else:
                # Перевірка, чи точки знаходяться на вертикальній лінії
                if self.x == other.x:
                    return EllipticCurvePoint(float('inf'), float('inf'), self.a, self.b, se

                # Обчислення коефіцієнта наклона прямої
                slope = ((other.y - self.y) * pow(other.x - self.x, -1, self.p)) % self.p

            # Обчислення координати x третьої точки
            x3 = (slope ** 2 - self.x - other.x) % self.p

            # Обчислення координати y третьої точки
            y3 = (slope * (self.x - x3) - self.y) % self.p

            # Повернення нової точки
            return EllipticCurvePoint(x3, y3, self.a, self.b, self.p)

        raise TypeError("Метод __add__ підтримує тільки об'єкти класу EllipticCurvePoint")

    def __str__(self):
        return f"({self.x}, {self.y})"


p = 23  # Поле простого числа
```

```
# Створення точок на еліптичній кривій
p1 = EllipticCurvePoint(3, 10, 1, 1, p)
p2 = EllipticCurvePoint(9, 7, 1, 1, p)

# Додавання точок
p3 = p1 + p2
print(f"p1 + p2 = ({p3.x}, {p3.y})")

# Порівняння точок
print(f"p1 == p2: {p1 == p2}")
print(f"p1 != p2: {p1 != p2}")

# Генерація координат для графіку еліптичної кривої
x = np.linspace(0, p - 1, 1000)
y = np.linspace(0, p - 1, 1000)
X, Y = np.meshgrid(x, y)
Z = (Y ** 2 - X ** 3 - p1.a * X - p1.b) % p

# Графік еліптичної кривої та точок
plt.contour(X, Y, Z, [0], colors='b')  # Еліптична крива
plt.plot(p1.x, p1.y, 'ro', label='P1')  # Точка P1
plt.plot(p2.x, p2.y, 'ro', label='P2')  # Точка P2
plt.plot(p3.x, p3.y, 'ro', label='P3')  # Точка P3

plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.title('Elliptic Curve')
plt.show()
```