Харківський національний університет імені В.Н. Каразіна

Факультет математики і інформатики

Кафедра прикладної математики

# Кваліфікаційна робота

**Магістр**

(освітньо-кваліфікаційний рівень)

на тему: Технологія блокчейн у Data Science: моделі токенів як інструмент нової фінансової системи

Виконала: студенка групи МП-62
ІІ курсу (магістерського рівня),
спеціальності 113 Прикладна математика

(шифр і назва напряму підготовки, спеціальності)

Прикладна математика

(освітньо-**професійна** програма)

Леонтьєва Катерина Сергіївна

Науковий керівник: доцент кафедри прикладної математики, кандидат фізико математичних наук

Стєпанова Катерина Вадимівна

Рецензент: доцент кафедри математичного моделювання та штучного інтелекту Національного аерокосмічного університету

кандидат технічних наук

Базілевич Ксенія Олексіївна

Харків - 2023 рік

# Анотація

У даній роботі ми розглянули сучасні алгоритми шифрування, а саме асиметричний алгоритм шифрування, який використовує математичні криві для шифрування даних, основи криптоаналізу щоб проаналізувати як здійснюються криптоатаки від шахраїв і як забезпечити захист даних за допомогою застосування аналітичного методу шифрування. Також зрозуміли що будь-яка криптосистема, заснована на дискретному логарифмуванні, може бути легко перенесена на еліптичні криві, тому ми перейшли до застосування модифікації систем Діффі-Хеллмана.

Ми визначили довжину побудови хеш-функції і зробили висновок, що у випадку, якщо функція $f$ не залежить від ключа, то для унеможливлення перебору коротких повідомлень вектор $H_0$ можна скласти з фрагментів, що вказують дату, час і т.п.

У другому розділі ми приділили багато уваги щодо Blockchain, а саме як виглядає структура блокчейну, навели приклад атомарного обміну криптовалют між уявними користувачами.

У третьому розділі ми навели приклади побудов графіків на еліптичних кривих і зробили реалізацію на еліптичній кривій у класичній формі Вейерштрасса.

# Annotation

In this work, we considered modern encryption algorithms, namely an asymmetric encryption algorithm that uses mathematical curves to encrypt data, the basics of cryptanalysis to analyze how crypto attacks are carried out by fraudsters and how to ensure data protection by applying an analytical method of encryption. We also realized that any cryptosystem based on discrete logarithms can be easily transferred to elliptic curves, so we moved on to applying the modification of Diffie-Hellman systems.

We determined the length of the construction of the hash function and concluded that if the function f does not depend on the key, then the $H_0$ vector can be composed of fragments indicating the date, time, etc., in order to prevent the sorting of short messages.

In the second chapter, we paid a lot of attention to Blockchain, namely what the blockchain structure looks like, and gave an example of an atomic exchange of cryptocurrencies between imaginary users.

In the third chapter, we gave examples of constructing graphs on elliptic curves and made an implementation on an elliptic curve in the classical Weierstrass form.

# ЗМІСТ

# ВСТУП

**Акутуальність.** Криптографія є дуже актуальною і важливою галуззю інформаційної безпеки в сучасному світі. Оскільки всі більші обсяги інформації зберігаються і передаються через мережі Інтернету, безпека цих даних стає дедалі важливішою. Криптографія забезпечує захист цих даних, щоб вони не потрапляли в руки неправильних людей, які можуть використовувати їх для злочинних цілей.

Крім того, криптографія є необхідною для безпеки важливих систем, таких як банківські системи, військові системи, системи зв'язку та інші. У цих системах криптографія забезпечує конфіденційність даних, що передаються, та автентифікацію користувачів, які мають доступ до системи.

Зростаюча популярність блокчейн-технологій і криптовалют також підвищує значимість криптографії. Блокчейн використовує криптографію для забезпечення безпеки даних, а криптовалюти використовують криптографію для забезпечення конфіденційності та безпеки транзакцій.

Із зростанням кількості кібератак та кіберзлочинності, захист інформації є все більш важливим.

Також криптографія використовується для забезпечення цифрової підпису, який вказує на те, що певний документ або повідомлення було підписано особою, яка має право на це. Це важливо для забезпечення довіри до електронних документів та операцій.

Крім того, поява квантових комп'ютерів може позначитися на актуальності криптографії. Ці комп'ютери можуть зламувати деякі криптографічні методи, які використовуються в даний час. Тому розробка нових криптографічних методів, що відповідають вимогам квантових комп'ютерів, стає все більш важливою.

Блокчейн технології є актуальними в сучасному світі і продовжують знаходити нові застосування в різних галузях діяльності. Вони забезпечують безпеку, прозорість та ефективну обробку даних, що робить

їх важливим інструментом для розвитку сучасного світу. Про розробки та плани використання технології блокчейн заявили платіжні системи VISA, Mastercard, Unionpay та SWIFT.

**Мета:** дослідження спрямоване на аналіз існуючих ресурсів з областей криптографії, блокчейну та еліптичних кривих. Мета полягає у розгляді цих джерел, доповненні представленого матеріалу власними прикладами та адаптації існуючих сценаріїв до мови програмування Python. Результатом роботи буде не лише систематизація інформації, але й надання конкретних ілюстрацій, а також їхнє втілення на практиці через програмний код на Python. Завершальною частиною будуть сформульовані висновки на основі проведеного аналізу та практичних застосувань, що дозволить висвітлити ключові аспекти досліджуваних тем.

# Розділ 1: Криптографія і блокчейн

## 1.1. Криптографія — наука про шифрування. Криптографія в блокчейн

Криптографія - це практика пересилання даних між двома чи більше сторонами при якому повідомлення є повністю зашифрованими та безпечними. Це дозволяє проводити конфіденційні транзакції з криптовалютою без залучення банківських посередників [2].

Криптографія відіграє важливу роль у блокчейні, розподіленій базі даних, що використовується для зберігання транзакцій та забезпечення безпеки у різних системах. Він забезпечує конфіденційність та цілісність інформації, а також захищає від несанкціонованого доступу та зміни даних.

Блокчейн використовує різні криптографічні методи, такі як асиметричне шифрування, хеш-функції, підписи та докази роботи [4].

Асиметричне шифрування використовується забезпечення конфіденційності в блокчейні. Воно дозволяє зашифровувати дані з використанням публічного ключа, доступного для всіх, і розшифровувати дані з використанням приватного ключа, яким володіє лише отримувач.

Хеш-функції використовуються для створення унікального «відбитка» даних, який неможливо змінити без зміни даних. Вони використовуються в блокчейні для створення блоків, які неможливо змінити після створення. Хеш-функції також використовуються для створення адрес гаманців у блокчейні, які неможливо зв’язати з конкретним користувачем без використання закритого ключа [14].

На сьогоднішній день існує широкий спектр сучасних алгоритмів шифрування, кожен з яких має свої унікальні особливості та переваги. Деякі з найпоширеніших алгоритмів шифрування включають:

1. **AES (Advanced Encryption Standard)** – це симетричний алгоритм шифрування, який використовується для захисту

конфіденційності даних у різних додатках, включаючи протоколи безпеки Wi-Fi та віртуальні приватні мережі (VPN).

2. **RSA (Rivest-Shamir-Adleman)** – це асиметричний алгоритм шифрування, який використовується для захисту конфіденційності повідомлень та даних. RSA часто використовується для створення цифрових підписів та безпеки електронної пошти.

3. **ECC (Elliptic Curve Cryptography)** – це асиметричний алгоритм шифрування, який використовує математичні криві для шифрування даних. ECC забезпечує більш високий рівень безпеки при використанні коротких ключів, ніж RSA.

4. **Blowfish** - це симетричний алгоритм шифрування, який застосовується для забезпечення конфіденційності даних у різних застосунках. Він має високу швидкодію шифрування та надійний захист від зламу.

5. **TLS (Transport Layer Security)** – це протокол шифрування, який використовується для захисту конфіденційності даних при передачі даних через Інтернет. TLS використовує різні алгоритми шифрування, включаючи AES и RSA, для забезпечення безпеки з'єднань.

Ці алгоритми шифрування застосовуються у різних застосунках та мережах з метою забезпечення безпеки та конфіденційності даних.

## 1.2. Блокчейн

Блокчейн представляє собою розподілену базу даних, організовану у вигляді ланцюжка блоків, які зв'язані за допомогою операцій хешування [4]. Структура даних ілюструється на наступному зображенні:

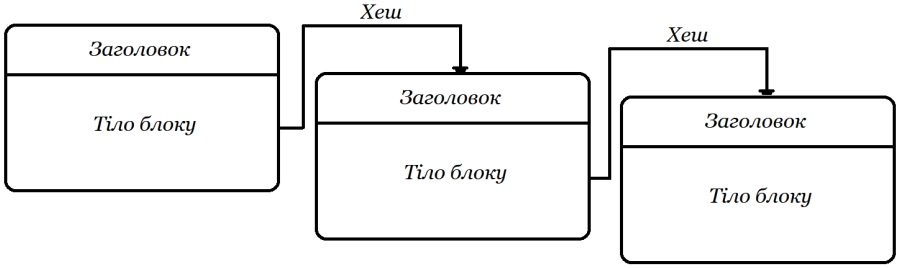


Рисунок 1.1 Структура блокчейну

Кожен блок в блокчейні складається з двох частин: корисної інформації, яка називається "тілом блоку", і службового заголовку, що виконує роль метаданих для організації даних. В заголовку кожного блоку міститься хеш-сума попереднього блоку. Такий підхід дозволяє використовувати хеш-значення як посилання для впорядкування блоків, подібно до алгоритмічної структури "зв'язний список". Головною перевагою цього підходу є зростаюча складність внесення змін у збережені дані.

Розглянемо наступну ситуацію: нехай зловмисник намагається підмінити дані блоку номер M, в блокчейні з N блоків. За умови зміни у інформації блоку, зміниться значення хеш-функції даного блоку(блоку M). За будовою блокчейну, так як хеш-сума блоку M записана у блоці M+1, зловмиснику доведеться записати нове значення хеш-суми у заголовок блоку $M + 1$, що вплине на його значення хеш-функції. Так, для підміни даних у блоці на глибині $h = N - M$, необхідно переробити обчислення для $h$ блоків. Для того, щоб було важко додати наступний блок, і тим самим унеможливити атаку на вже записані блоки, використовують спеціальні техніки розподіленого консенсусу. Таким чином, з появою кожного наступного блоку, кількість роботи, яку необхідно проробити для модифікації конкретного блоку, зростає. Це дозволяє сприймати інформацію в блоках, починаючи з певної глибини, як істинну.

Криптографічний алгоритм, який використовується в біткойні та блокчейні, базується на дискретному логарифмі для еліптичних кривих на скінченних полях, який подібний до дискретного логарифму в скінченному полі [3]. Таким чином, якщо розглядати еліптичну криву c на скінченному полі Fp, то задача дискретного логарифмування полягає в тому, щоб за двома точками $P, Q \in c$ знайти, чи існує $n \in \mathbb{Z}^+$ таке, що $nP = Q$.

З дискретним логарифмом на еліптичних кривих ми отримуємо нову математичну задачу, яка є простою з одного боку, але дуже складною з іншого. У результаті, враховуючи $n \in \mathbb{Z}^+$ і $P \in c$, обчислення $Q = nP$ є відносно простим. З іншого боку, враховуючи отримання дискретного логарифма, тобто $n$, є дуже складною проблемою, коли $n$ велике.

Складність операції сумування на еліптичній кривій вказує на те, що дискретний логарифм на еліптичній кривій є набагато складнішим, ніж відповідний для скінченних полів.

## 1.3. Мережа Bitcoin

Bitcoin являє собою першу реалізацію криптовалюти, що використовувала блокчейн. Кожен блок складається з заголовку, що містить службову інформацію та тіла, що складається зі списку транзакцій. Для спрощення верифікації конкретних транзакцій, заголовок блоку містить хеш кореня дерева Меркле, що побудоване на списку транзакцій блоку [14].

**Цифровий підпис.** Для підписання транзакцій використовується асиметрична криптографія з використанням еліптичних кривих, зокрема використовується еліптична крива secp256k1 та алгоритм ECDSA. Дана крива визначена на скінченному полі і дозволяє ефективно проводити обчислення.

**Адреси.** Для виконання операцій, кожен учасник мережі створює аккаунт, який може мати необмежену кількість пар ключів. Ідентифікатором у мережі Біткойн, що дозволяє створювати транзакції слугує адреса. Адреса визначається з публічного ключу, за допомогою застосування хеш-функції RIPEMD-160.

Так, адреса має розмір 20 байт і найчастіше записується за допомогою кодування base58 (для запобігання використання не розрізнювальних літер у вигляді 34 символів латинського алфавіту.

**Смарт-контракти.** Для контролю за правом розпоряджатись токенами, використовується концепція блокуючих та розблокуючих

скриптів мовою Bitcoin Script. Мова складається з невеликої кількості операційних кодів та не є Тьюринг-повною, адже у ній відсутні цикли. Інтерпретатор даної мови використовує стекову модель.

**Транзакції.** Структуру транзакції задає набір входів транзакції, набір виходів транзакції, та певна службова інформація. Транзакції бувають різних видів і класифікуються в залежності від умов, що потрібно виконати, аби витрати кошти, що переказуються. Так, класичною транзакцією є P2PKH - тобто переказ токенів на задану адресу. Для реалізації більш складних технік контролю доступу, використовуються транзакції типу P2SH, що дозволяють використання довільних блокуючих скриптів.

**Баланс.** Баланс кожної адреси не зберігається у блокчейні, а перераховується за запитом з публічної інформації. Кількість токенів, що належить певній адресі визначається сумарним об’ємом невитрачених виходів транзакцій(UTXO), у яких дана адреса вказана в якості отримувача.

# Розділ 2: Дискретне логарифмування, система Діффі-Хеллмана

## 2.1. Задача дискретного логарифмування

Розглянемо методи розв'язування задачі дискретного логарифмування $h = g^x$ у різних групах $G$ [8]. Ці алгоритми належать до однієї з двох категорій: або алгоритми є загальними та застосовуються до будь-якої скінченної абелевої групи, або алгоритми специфічні для спеціальної групи, що розглядається. Почнемо з алгоритмів загального призначення.

**Поліг–Хеллман**

Перше зауваження, яке слід зробити, полягає в тому, що проблема дискретного логарифмування в групі G настільки ж складна, як і проблема дискретного логарифмування в найбільшій підгрупі простого порядку в $G$. Це спостереження зроблено Полігом і Хеллманом, і воно застосовне до довільної скінченної абелевої групи [8].

Припустимо, що ми маємо скінченну циклічну абелеву групу $G = \langle g \rangle$, порядок якої задається формулою

$$N = \prod_{i=1}^{t} p_i^{e_i}.$$

Тепер припустимо, що задано $h \in G$, таке, що існує ціле число $x$, таке, що

$$h = g^x.$$

Наша мета — знайти $x$, спочатку знайшовши його за модулями $p_i^{e_i}$, а потім використовуючи китайську теорему про остачі відновити його за модулем $N$.

З теорії груп відомо, що існує груповий ізоморфізм

$$\varphi: G \to C_{p_1^{e_1}} \times \cdots \times C_{p_t^{e_t}},$$

де $C_{p^e}$ — циклічна група порядку $p^e$. Проекція φ на компонент $C_{p^e}$ задається як

$$\varphi_p = \{G \ -> C_{p^e}, f \ -> f^{N/p^e}\}.$$

Відображення $\varphi_p$ є груповим гомоморфізмом [8], отже, якщо ми маємо $h = g^x$ у $G$, то ми матимемо $\varphi_p(g)^x$ у $C_{p^e}$. Але дискретний логарифм у $C_{p^e}$ визначається лише за модулем $p^e$. Отже, якби ми могли розв'язати задачу дискретного логарифмування в $C_{p^e}$, ми б визначили $x$ за модулем $p^e$. Зробивши це для всіх простих чисел $p$, що ділять $N$, ми маємо змогу розв'язати x за допомогою китайської теореми про остачі.

Припустимо, що ми маємо якийсь оракул $O(g, h, p, e)$, який для $g, h \in C_{p^e}$ виведе дискретний логарифм $h$ відносно $g$. Потім ми можемо знайти $x$ за допомогою наступного алгоритму [8].

Алгоритм (псевдокод):

S = {}

Для всіх простих $p$, що ділять $N$:

Обчислити найбільше $e$ таке, що $T = p^e$ ділить $N$

$g_1 = g^{N/T}$

$h_1 = h^{N/T}$

$z = O(g1, h1, p, e)$

$S = S + \{(z, T)\}$

$x = CRT(S)$

Тепер необхідно показати, як вирішити проблему дискретного логарифмування в $C_{p^e}$. Ця задача зводиться до розв'язування задач дискретного логарифмування в групі $C_p$ [8]. Припустимо, що $g, h \in C_{p^e}$ і існує такий $x$, що $h = g^x$.

Очевидно, що $x$ визначено лише за модулем $p^e$, і ми можемо записати

$$x = x_0 + x_1 p + \cdots + x_{e-1} p^{e-1}.$$

Знаходимо $x_0, x_1, \ldots$ по черзі, використовуючи таку індуктивну процедуру. Припустимо, що ми знаємо $x'$, значення $x$ по модулю $p^t$, тобто

$$x' = x_0 + \cdots + x_{t-1} p^{t-1}.$$

Тепер ми хочемо визначити $x_t$ і таким чином обчислити $x$ за модулем $p^{t+1}$. Запишемо

$$x = x' + p^t x'',$$

тоді маємо

$$h = g^x (g^{p^t})^{x''},$$

отже, якщо ми позначимо

$$h' = h g^{-x'} \text{ і } g' = g^{p^i}$$

то

$$h' = g'^{x''}.$$

Тепер $g'$ є елементом порядку $p^{e-t}$, тому щоб отримати елемент порядку $p$, а отже, дискретний логарифм задачі в $C_p$ нам потрібно піднести наведене вище рівняння до степеня $s = p^{e-t-1}$. Тому позначаючи

$$h'' = h'^s \text{ і } g'' = g'^s$$

ми отримуємо задачу дискретного логарифмування в $C_p$, задану як

$$h'' = g''^{x_1}.$$

Отже, припускаючи, що ми можемо розв’язати дискретні логарифми в $C_p$, ми можемо знайти $x_t$ і, отже, знайти $x$ [8].

**Метод Baby-Step/Giant-Step**

У нашому обговоренні алгоритму Поліга–Хелмана ми припустили, що маємо оракул для вирішення проблеми дискретного логарифмування в циклічних групах простого порядку. Тепер ми опишемо загальний метод розв’язування таких задач, який називається методом Baby-Step/Giant-Step [8]. Знову ж таки, це загальний метод, який застосовується до будь-якої скінченної циклічної абелевої групи.

Оскільки проміжні кроки в алгоритмі Поліга–Хеллмана досить прості, складність розв'язання загальної задачі дискретного логарифмування буде домінувати через час, необхідний для вирішення задачі дискретного логарифмування в циклічних підгрупах простого порядку.

Отже, для загальних груп складність методу Baby-Step/Giant-Step домінуватиме над загальною складністю будь-якого алгоритму. Дійсно, можна показати, що наступний метод є найкращим із можливих методів, з точки зору часу, для вирішення проблеми дискретного логарифмування в довільній групі. Звичайно, у будь-якій фактичній групі може бути алгоритм спеціального призначення, який працює швидше, але загалом наступне є найкращим, що можна зробити [8].

Позначаємо відкриту циклічну групу як $G = \langle g \rangle$, яку ми тепер можемо вважати простим порядком p. Нам також задано $h \in G$, і необхідно знайти ціле число $x$, таке, що

$$h = g^x \bmod p.$$

Ми припускаємо, що існує певне фіксоване кодування елементів $G$, тому, зокрема, легко зберігати, сортувати та шукати список елементів $G$ [8].

Ми спочатку пишемо

$$x = x_0 + x_1[\sqrt{p}].$$

Тепер, оскільки $x \leq p$, ми маємо, що $0 \leq x_0, x_1 < \lceil\sqrt{p}\rceil$.

Спочатку ми обчислюємо Baby-Steps

$$g_i = g^i \text{ для } 0 \leq i < \lceil\sqrt{p}\rceil.$$

Пари

$$(g_i, i)$$

зберігаються в таблиці, щоб можна було легко шукати елементи, індексовані першим записом у парі. Це може бути досягнуто шляхом сортування таблиці за першим записом або ефективніше за допомогою хеш-таблиць. Щоб обчислювати та зберегти Baby-Steps, потрібно $O(\lceil\sqrt{p}\rceil)$ часу і аналогічний обсяг зберігання [8].

Тепер ми обчислюємо Giant-Steps

$$h_j = hg^{-j\lceil\sqrt{p}\rceil} \text{для } 0 \leq j < \lceil\sqrt{p}\rceil.$$

Далі ми намагаємося знайти відповідність у таблиці Baby-Steps, тобто ми намагаємося знайти таке значення $g_i$, що $g_i = h_j$. Якщо співпадіння відбудеться, ми маємо

$$x_0 = i \text{ та } x_1 = j$$

оскільки, якщо $g_i = h_j$,

$$g^i = hg^{-j\lceil\sqrt{p}\rceil},$$

тобто

$$g^{i+j\lceil\sqrt{p}\rceil} = h.$$

Зауважимо, що час для обчислення Giant-Step становить щонайбільше $O(\lceil\sqrt{p}\rceil)$. Отже, загальна часова та просторова складність методу Baby-Step/Giant-Step становить $O(\lceil\sqrt{p}\rceil)$.

У поєднанні з алгоритмом Поліга–Хеллмана це означає, що якщо ми хочемо, щоб проблема дискретного логарифмування в групі $G$ була складності в $2^{80}$ операцій, то нам потрібно, щоб група $G$ мала підгрупу простого порядку з порядком більше ніж $2^{160}$.

**Приклад**

Як приклад розглянемо підгрупу порядку 101 у мультиплікативній групі скінченного поля $\mathbb{F}_{607}$, породжену $g = 64$. Припустимо, що нам задано задачу дискретного логарифмування

$$h = 182 = 64^{x} \, mod \, 607.$$

Спочатку ми обчислюємо Baby-Steps

$$g_i = 64^{i} \, mod \, 607 \text{ для } 0 \leq i < \lceil\sqrt{101}\rceil = 11.$$

Ми обчислюємо

| $i$ | $64^{i} \, mod \, 607$ | $i$ | $64^{i} \, mod \, 607$ |
|---|---|---|---|
| 0 | 1 | 6 | 330 |
| 1 | 64 | 7 | 482 |
| 2 | 454 | 8 | 498 |
| 3 | 527 | 9 | 308 |
| 4 | 343 | 10 | 288 |

| | | | |
|---|---|---|---|
| 5 | 100 | | |

Тепер ми обчислюємо гігантські кроки,

$$h_j = 182 \cdot 64^{-11j} \, mod \, 607 \text{ для } 0 \leq j < 11,$$

і перевіряємо, коли ми отримаємо Giant-Step, який зустрічається в нашій таблиці Baby-Steps:

| $j$ | $182 \cdot 64^{-11j} \, mod \, 607$ | $j$ | $182 \cdot 64^{-11j} \, mod \, 607$ |
|---|---|---|---|
| 0 | 182 | 6 | 60 |
| 1 | 143 | 7 | 394 |
| 2 | 69 | 8 | 483 |
| 3 | 271 | 9 | 76 |
| 4 | 343 | 10 | 580 |
| 5 | 573 | | |

Отже, ми отримуємо збіг, коли $i = 4$ і $j = 4$, що означає, що $x = 4 + 11 \cdot 4 = 48$, ми можемо підтвердити, що це правильна відповідь на попередню задачу дискретного логарифмування, обчисливши

$$64^{48} (mod \, 607) = 182.$$

Отже, через наявність алгоритму Поліга–Хеллмана задачу дискретного логарифмування слід ставити в групі, порядок якої має великий простий множник.

Алгоритм Baby-Step/Giant-Step — це загальний алгоритм, час роботи якого може бути абсолютно обмежений $\sqrt{q}$, де $q$— розмір великого простого множника $\#G$. Однак вимоги до зберігання алгоритму Baby-Step/Giant-Step також становлять $O(\sqrt{q})$ [8].

Зауважимо, що існує ряд методів, завдяки Полларду, заснованих на детермінованих випадкових блуканнях в групі. Це загальні алгоритми, які потребують мало пам'яті, але вирішують проблему дискретного логарифмування за очікуваний час $O(\sqrt{q})$. Одним із таких алгоритмів є алгоритм Pollard's Rho.

## 2.2. Система Діффі-Хеллмана

Проблема розподілу ключів була вирішена в статті Діффі та Хеллмана, в якому вони представили криптографію з відкритим ключем. Їх протокол розподілу ключів називається "Diffie–Hellman Key Exchange" і дозволяє двом сторонам погодити секретний ключ через незахищений канал без попередньої зустрічі [8]. Його безпека базується на задачі дискретного логарифмування в скінченній абелевій групі $G$.

В оригінальній статті група прийнята як $G = F^*_p$, (зараз існують більш ефективні версії отримані шляхом прийняття G як групи еліптичних кривих, де протокол називається EC-DH).

У обох сторін є свої власні ефемерні секрети $a$ і $b$. З цьома обидві сторони можна узгодити той самий секретний ключ сеансу:

- Аліса може обчислити $K = (g^b)^a \, mod \, p$, оскільки вона знає $a$ і Боб надіслав їй $g^b$,
- Боб також може обчислити $K = (g^a)^b \, mod \, p$, оскільки він знає $b$ і Аліса надіслала йому $g^a$.

Зловмисник може бачити повідомлення $g^a$ і $g^b$ і має відновити секретний ключ

$$K = g^{ab},$$

Тобто розв'язати задачу дискретного логарифмування, яка є практично нерозв'язною при великих значеннях $p$ (найшвидший відомий алгоритм має суб експоненціальну складність).

**Приклад (python) Генерація ключів та зламування методом Pollard-ρ.**

```
from sympy import nextprime, primitive_root, gcd
from egcd import egcd

def f(x, a, b, g, h, n):
    x1 = x
    if x1 % 3 == 0:
        return (x**2) % n, (2 * a) % (n - 1), (2 * b) % (n -
1)
    elif x1 % 3 == 1:
        return (g * x) % n, (a + 1) % (n - 1), b
    elif x1 % 3 == 2:
        return (h * x) % n, a, (b + 1) % (n - 1)

def mydLog(g, h, p):
    xi, ai, bi = 1, 0, 0
    x2i, a2i, b2i = 1, 0, 0

    while True:
        xi, ai, bi = f(xi, ai, bi, g, h, p)
        xi, ai, bi = f(xi, ai, bi, g, h, p)
        x2i, a2i, b2i = f(x2i, a2i, b2i, g, h, p)
        if xi == x2i:
            break

    a = ai - a2i
    b = b2i - bi
    d = gcd(b, p - 1)

    if d == 1:
        _, alpha, beta = egcd(b, p - 1)
        return (alpha * a) % (p - 1)
```

```
    _, s, t = egcd(b, p - 1)
    i = 0

    while i < d:
        y = int((a * s + i * (p - 1)) // d)
        if pow(g, y, p) == h:
            return y % (p - 1)
        i += 1

p1 = nextprime(2**37 + 1)
g = primitive_root(p1)
Rpub = 114534287914
Dpub = 47963704417

a = mydLog(g, Rpub, p1)
print(a)

s = pow(Dpub, a, p1)
print(s)
```

Зауважимо, що дискретні логарифми складно обчислюються, коли число $p-1$ містить один великий простий множник, наприклад, коли його можна представити у вигляді $p-1=2q$, де q — просте число. Припустимо, що просте число $p=2q+1$ обрано. Наступним кроком потрібно вибрати первісний корінь $g$ за модулем $p$. Для цього можна скористатися критерієм, згідно з яким число $g$ буде первісним коренем за модулем p тоді і тільки тоді, коли: $g^2 \neq 1(\mathrm{mod}\, p)$, $g^q \neq 1(\mathrm{mod}\, p)$.

## 2.3. Модифікація системи Діффі-Хеллмана на еліптичних кривих

Безпека криптосистем на еліптичних кривих ECC (Elliptic Curve Cryptography) як правило, заснована на труднощі вирішення завдання **дискретного логарифмування в групі точок еліптичної кривої.**

В класі криптосистем з відкритим ключем криптосистеми на еліптичних кривих перевершують класичні криптосистеми на основі модулярної арифметики, як мінімум, за двома важливими параметрами:

ступенем захищеності у розрахунку на кожен біт ключа та швидкодію при апаратній та програмній реалізації [8].

Будь-яка криптосистема, заснована на дискретному логарифмуванні, може бути легко перенесена на еліптичні криві.

Тут операція $y = g^x (\mathrm{mod}\, p)$ замінюється на $Y = [x]G$. Нехай $q$ - деякий (досить великий) простий дільник числа $|E_p(a,\ b)|$ та деяка точка $G \in E_p(a, b)$ має порядок q, тобто. утворює циклічну підгрупу порядку q в $(E_p(a, b), +)$ :

$\langle G \rangle = \{G,\ [2]G,\ \dots\ ,\ [q]G = \mathcal{O}\}$.

Загальнодоступні параметри системи: $p, q, G, E_p(a, b)$. Абоненти А, В, … , вибирають відповідні секретні ключі $x_A, x_B$ і т.д., що не перевищують числа $q - 1$. По кожному секретному ключу обчислюється відкритий ключ:

$Y_A = [x_A]G$,

$Y_B = [x_B]G$,

… ,

які розміщуються у загальнодоступному довіднику разом із параметрами системи. Якщо абоненти А та В хочуть організувати секретний зв'язок, то абонент А обчислює значення [8]:

$t_{BA} = [x_A]Y_B$,

а абонент В обчислює:

$t_{AB} = [x_B]Y_A$,

При цьому $T_{AB} = T_{BA} = [x_A x_B]G$. Тепер абоненти А і В можуть використовувати, наприклад, абсцису точки $T_{AB}$ як ключ для секретного листування.

# Розділ 3. Реалізація еліптичних кривих

## 3.1. Еліптичні криві

Криптографія еліптичної кривої (ECC) — це сучасна сімейство криптосистем з відкритим ключем, яка базується на алгебраїчних структурах еліптичних кривих над кінцевими полями та на складності проблеми дискретного логарифму еліптичної кривої (ECDLP).

У математиці еліптичні криві — це плоскі алгебраїчні криві, що складаються з усіх точок {x, y}, що описуються рівнянням [14]:

$$Ax^3 + Bx^2y + Cxy^2 + Dy^3 + Ex^2 + Fxy + Gy^2 + Hx + Iy + J = 0$$

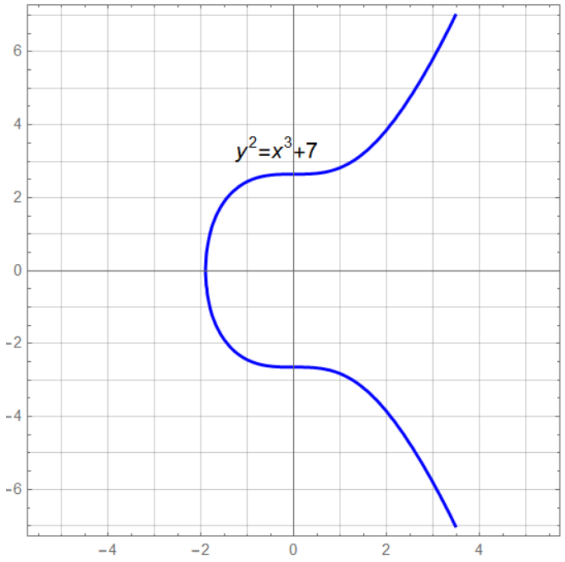
Криптографія використовує еліптичні криві у спрощеній формі (форма Вейєрстраса), яка визначається як:

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

Наприклад, крива NIST secp256k1 (використовується в Bitcoin) базується на еліптичній кривій у формі:

$$y^2 = x^3 + 7$$

(наведене вище рівняння еліптичної кривої, де a = 0 і b = 7). Це візуалізація наведеної вище простої еліптичної кривої:




Криптографія еліптичної кривої (ECC) використовує еліптичні криві над скінченним полем $\mathbb{F}_p$ (де p є простим числом і p > 3) або $\mathbb{F}_{2m}$ (де розмір

полів $p = 2m$) [14]. Це означає, що поле є квадратною матрицею розміром $p \times p$, а точки на кривій обмежені лише цілими координатами в межах поля. Усі алгебраїчні операції в полі (як-от додавання та множення точок) призводять до іншої точки в полі. Рівняння еліптичної кривої над скінченним полем $\mathbb{F}_p$ набуває наступного модульного вигляду:

$$y^2 \equiv x^3 + ax + b \text{ (mod p)}.$$

Відповідно, «крива біткойна» secp256k1 набуває вигляду:

$$y^2 \equiv x^3 + 7 \text{ (mod p)}$$

На відміну від RSA, який використовує для свого ключового простору цілі числа в діапазоні [0...p-1] (поле $\mathbb{Z}_p$), ECC використовує точки {x, y} у межах поля Галуа $\mathbb{F}_p$ (де x і y є цілі числа в діапазоні [0...p-1]).

Еліптична крива над скінченним полем $\mathbb{F}_p$ складається з набору цілих координат {x, y}, таких що $0 \leq x, y < p$ залишаються на еліптичній кривій: $y^2 \equiv x^3 + ax + b \ (mod\ p)$ [14].

Приклад еліптичної кривої над скінченним полем $\mathbb{F}_{17}$:

еліптична крива над скінченним полем $y^2 \equiv x^3 + 7 \ (mod\ 17)$ складається з фіолетових точок на наведеному нижче малюнку, тобто на практиці «еліптичні криві», які використовуються в криптографії, є «набором точок у квадратній матриці», а не класичними. «кривими».

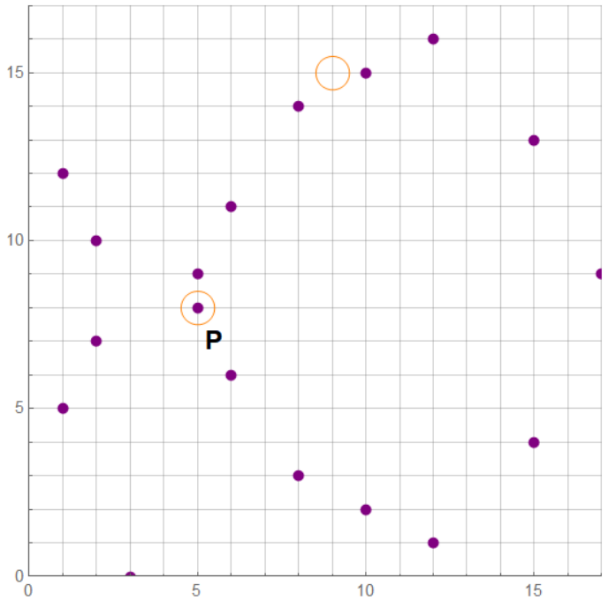
Наведена вище крива є «навчальною». Він забезпечує дуже малу довжину ключа (4-5 біт). У реальному світі розробники зазвичай використовують криві 256-біт або більше [14].

Досить легко обчислити, чи належить певна точка певній еліптичній кривій над скінченним полем. Наприклад, точка {x, y} належить кривій $y^2 \equiv x^3 + 7 \pmod{17}$ тоді і тільки тоді, коли:

$$x^3 + 7 - y^2 \equiv (mod\ 17)$$

Точка P {5, 8} належить кривій, оскільки (5**3 + 7 - 8**2) % 17 == 0. Точка {9, 15} не належить кривій, оскільки (9 **3 + 7 - 15**2) % 17 != 0. Ці обчислення виконано в стилі Python.

Еліптична крива та точки {5, 8} і {9, 15} візуалізуються на наступному графіку:




## 3.2. Дискретне логарифмування для еліптичних кривих

Для еліптичних кривих немає відомих субекспоненціальних методів для проблеми дискретного логарифмування, за винятком деяких особливих випадків [10]. Це означає, що єдиним методом в цьому налаштуванні є паралельна версія методу Ро Полларда [8].

Нехай еліптична крива $E$ задана над скінченним полем $\mathbb{F}_q$. Позначимо

$$\#E(\mathbb{F}_q) = h \cdot r$$

де $r$ – просте число. За теоремою Хассе значення $\#E(\mathbb{F}_q)$ близьке до $q$, тому ми зазвичай вибираємо криву з $r$, близьким до $q$, тобто ми вибираємо криву E так, щоб $h = 1, 2$ або $4$.

Найвідомішим загальним алгоритмом для задачі дискретного логарифмування еліптичної кривої є паралельний Метод Ро Полларда, який має складність $O(\sqrt{r})$, що становить приблизно $O(q)$. Отже, для досягнення такої ж безпеки, як і 80-бітний блоковий шифр, нам потрібно взяти $q \approx 2^{160}$ [8].

Однак існує ряд особливих випадків, яких слід уникати. Розглянемо ці особливі випадки, але ми не будемо наводити детальні причини, чому їх

слід уникати. Як зазвичай, ми припускаємо, що $q$ є або великим простим числом, або степенем двійки.

• Для будь-якого $q$ ми повинні вибрати криві, для яких не існує малого числа $t$ такого, що $r$ ділиться $qt - 1$, де $r$ є великим простим множником $\#E(\mathbb{F}_q)$. Це усуває вироджені криві та деякі інші. У цьому випадку існую просте обчислюване відображення з задачі дискретного логарифмування для еліптичної кривої до задачі дискретного логарифмування в кінцевому поле $\mathbb{F}_{q^t}$ . Отже, в цьому випадку ми отримуємо субекспоненціальний метод розв'язування дискретного логарифмування для еліптичної кривої [8].

• Якщо $q = p$ є великим простим числом, тоді нам потрібно уникати аномальних кривих, де $E(\mathbb{F}_p) = p$. В цьому випадку є алгоритм, який вимагає $O(log\ p)$ операцій еліптичної кривої.

• Якщо $q = 2n$, ми зазвичай припускаємо, що n є простим числом, щоб уникнути можливості певних атак на основі концепції «спуску Вейля» [13].

## 3.3. Реалізація мовою Python

**Приклад: множення EC Point на ціле число**

Формули множення EC відрізняються для різних форм представлення кривої. У цьому прикладі ми будемо використовувати еліптичну криву в класичній формі Вейєрштрасса.

Наприклад, візьмемо точку EC G = {15, 13} на еліптичній кривій над кінцевим полем $y^2 \equiv x^3 + 7\ (mod\ 17)$ і помножимо її на $k = 6$. Отримаємо точку EC P = {5, 8 }:

$$P = k * G = 6 * \{15, 13\} = \{5, 8\}$$

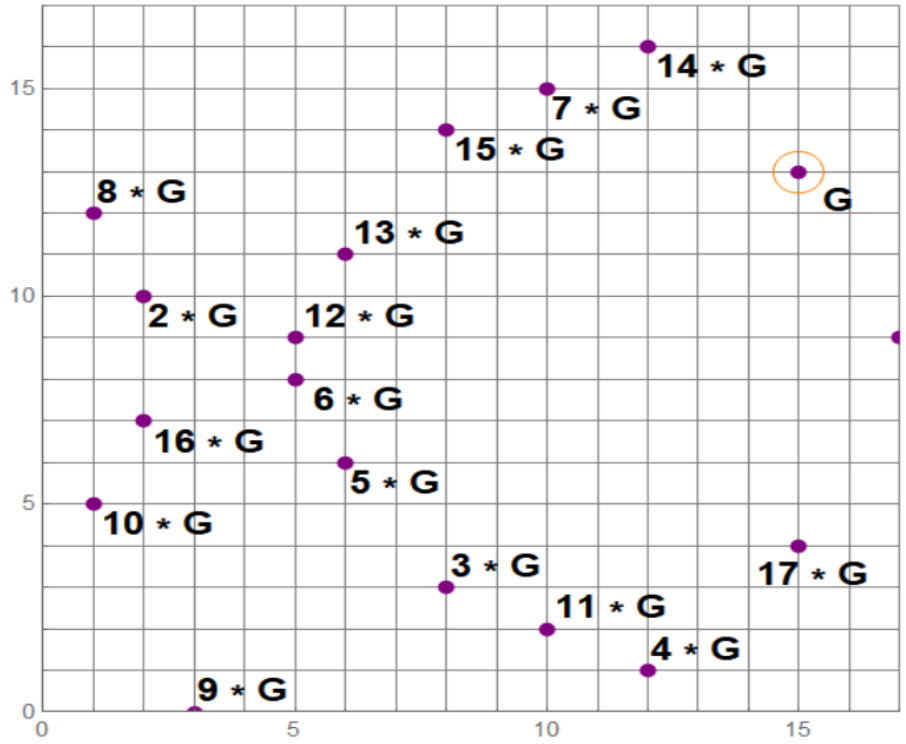
Ми працюємо з навчальною кривою з наших попередніх прикладів $y^2 \equiv x^3 + 7\ (mod\ 17)$, з твірною точкою G = {15, 13}, яка має порядок $n = 18$ [14].

```
pip install tinyec
import tinyec.ec as ec
# Define the parameters for the subgroup
field_params = {'p': 17, 'g': (15, 13), 'n': 18, 'h': 1}
subgroup = ec.SubGroup(**field_params)

# Create the elliptic curve using the subgroup parameters
curve = ec.Curve(a=0, b=7, field=subgroup)

# Calculate and print k * G for each k in the range [0, 24]
for k in range(25):
    point_k_times_G = k * curve.g
    print(f"{k} * G = ({point_k_times_G.x}, {point_k_times_G.y})")
```

Код вище демонструє множення EC. Він множить генераторную точку G на 0, 1, 2, ..., 24. Вихід із наведеної вище програми такий:

```
0 * G = (None, None)
1 * G = (15, 13)
2 * G = (2, 10)
3 * G = (8, 3)
4 * G = (12, 1)
5 * G = (6, 6)
6 * G = (5, 8)
7 * G = (10, 15)
8 * G = (1, 12)
9 * G = (3, 0)
10 * G = (1, 5)
11 * G = (10, 2)
12 * G = (5, 9)
13 * G = (6, 11)
14 * G = (12, 16)
15 * G = (8, 14)
16 * G = (2, 7)
17 * G = (15, 4)
18 * G = (None, None)
19 * G = (15, 13)
20 * G = (2, 10)
21 * G = (8, 3)
22 * G = (12, 1)
23 * G = (6, 6)
24 * G = (5, 8)
```

Видно, що $0 * G = \infty$. Також добре видно, що група еліптичної кривої є циклічною і порядок групи $n = 18$, оскільки починаючи з




$k = 18$ наступні точки повторюють перші.

Трохи модифікуємо наведений вище приклад і змінимо точку генератора на G' = {5, 9} [14]. Це значно змінить результат:

```
import tinyec.ec as ec

# Define the parameters for the subgroup
field_params = {'p': 17, 'g': (5, 9), 'n': 18, 'h': 1}
subgroup = ec.SubGroup(**field_params)

# Create the elliptic curve using the subgroup parameters
curve = ec.Curve(a=0, b=7, field=subgroup)

# Calculate and print k * G for each k in the range [0, 24]
for k in range(25):
    point_k_times_G = k * curve.g
    print(f"{k} * G = ({point_k_times_G.x}, {point_k_times_G.y})")
```

Результат:

```
0 * G = (None, None)
1 * G = (5, 9)
2 * G = (5, 8)
3 * G = (None, None)
4 * G = (5, 9)
```

```
5 * G = (5, 8)
6 * G = (None, None)
7 * G = (5, 9)
8 * G = (5, 8)
9 * G = (None, None)
10 * G = (5, 9)
11 * G = (5, 8)
12 * G = (None, None)
13 * G = (5, 9)
14 * G = (5, 8)
15 * G = (None, None)
16 * G = (5, 9)
17 * G = (5, 8)
18 * G = (None, None)
19 * G = (5, 9)
20 * G = (5, 8)
21 * G = (None, None)
22 * G = (5, 9)
23 * G = (5, 8)
24 * G = (None, None)
```

Розглянемо тепер реальний приклад. Замість використання навчальної кривої (4-5-бітна крива, p = 17), ми будемо використовувати 192-бітну криптографічну криву secp192r1 (192-біт, p = 6277101735386680763835789423207666416083908700390324961279). Наведений нижче приклад схожий на попередній [14]:

```python
import tinyec.ec as ec

# Retrieve the curve parameters for secp192r1 from the registry
curve = ec.curve = registry.get_curve('secp192r1')

# Calculate and print k * G for each k in the range [0, 24]
for k in range(25):
    point_k_times_G = k * curve.g
    print(f"{k} * G = ({point_k_times_G.x}, {point_k_times_G.y})")
print(f"n * G = ({nG.x}, {nG.y})")
```

Результат:

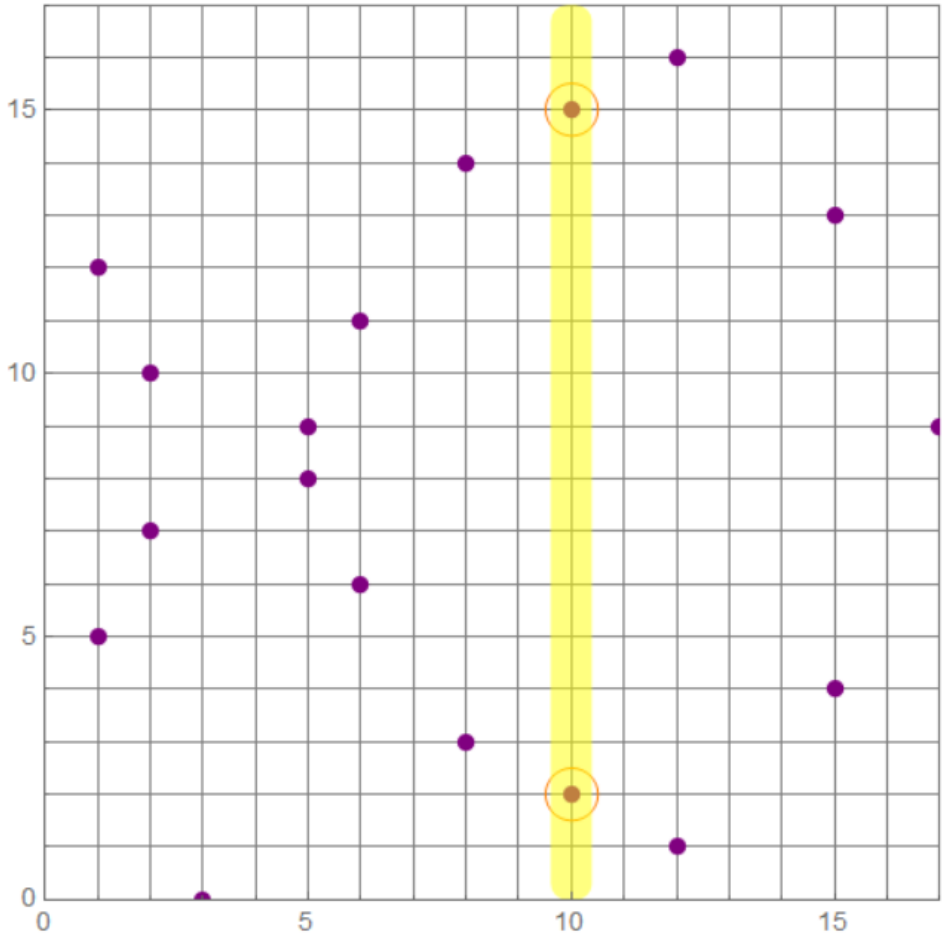
```
0 * G = (None, None)
1 * G = (6020462823756886567582134805875261119166989766368846848 18,
1740503322936220314048575522802194103640234889273866506 41)
```

```
2 * G = (53697444036787105634324583612545441709660963845867644294 48,
5429234379789071039750654906915254128254326554272718558123)
3 * G = (29151096302806788907202067797069634555906274658861031351 94,
2946626711558792003980654088990112021985937607003425539581)
4 * G = (13059948804309039973059437386977794083169295652347878371 14,
3981863977451150342116987835776121688410789618551673306674)
5 * G = (41028325111678487401899356213656687011067670693676266024 0,
1206654674899825246688205669651974202006189255452737318561)
6 * G = (40085041464535260251731969003035941557999956279102318999 46,
3263759301305176906990806636587838100022690095020155627760)
7 * G = (34733390813784061238528712993952624762896724797070383505 89,
2152713176906603604200842901176476029776544337891569565621)
8 * G = (11679506110148945123130333626966974414973400813908414909 10,
4002177906111215127148483369584652296488769677804145538752)
9 * G = (31763174504537056502837758112284936267764894333096364750 23,
4460189377466938476679380385498011517961211807501706220 1)
10 * G = (41802945013483680838095632350213700573755914059309928032 05,
1227781623738814009517798297176766391967714436501424281520)
11 * G = (70124600887874588195510636281397533591347599456631368157 8,
5862504000586036904137784111819119357991480972474652990468)
12 * G = (40162916061236298848064820758772324784742702617147727684 3,
2207396948372169770500650183616837914516487320984174930239)
13 * G = (42095185291891953981961283437605569829547417806681525684 2,
2698412698688357409961393305195305773566297426026390297630)
14 * G = (48361646636492811454120688375909600467559433105920494841 7,
7567573688143511018108841809711794557742861008392897757 96)
15 * G = (34471174602664997603883760874383569791900756171309863858 95,
9420500640415137854577959946285312414709478216007700827 37)
16 * G = (44918444280981149878772372484889696169931290655133401538 23,
6267780533505741098894309136881239254479487913874120712848)
17 * G = (16711253358348278060915581501697841671343502986556002138 98,
5876882175903236860388081433302259871828398972419132219344)
18 * G = (47496687458027917326845819784949245712854823056959784595 55,
7393043311499112879896915833998872934595401588540097493 31)
19 * G = (47172531729198341191470797062805913940715844521364779614 99,
3846238242484366591443667019582044776719878905617448608266)
20 * G = (45958613905865160532273328924070619804079497694161788511 75,
3577310604553808740271041971695937863431408727974102387853)
21 * G = (55047906916360634618512407456733309233088979375897434461 03,
3202719333168318922810922862026902890711656209755282037 80)
22 * G = (24597500196220474257396157037402832588286689413460022155 89,
1899746235467922133432526115687374784718405851179564643213)
23 * G = (35214239829885811783761462698746785323894794649185125265 42,
2074488862549995765951423029198249412082089608186357559 88)
24 * G = (62098448269476047900389750562672081462580252685195124176 42,
2994524308329009013576298176683420492475513562601889388197)
n * G = (None, None)
```

Крива secp192r1 використовує циклічну групу дуже великого порядку n (просте число) з співмножником $h = 1$, і, як ми можемо очікувати, $n * G = \infty$, як і в попередньому прикладі.

## 3.4. Стиснення відкритих ключів у криптосистемах з еліптичним ключем

Еліптичні криві над скінченними полями $\mathbb{F}_p$ (у формі Вейєрштрасса) мають щонайбільше 2 точки на координату $y$ (непарні $x$ і парні $x$) [14]. Ця властивість походить від природи рівняння еліптичної кривої та проілюстрована на графіку нижче:




Завдяки цій властивості точку еліптичної кривої (і відповідно відкритий ключ ECC) P {x, y} можна стиснути як C {x, непарний/парний). Це означає стерти координату y з точки та представити її як 1 біт (непарне y або парне y) [14].

Стиснута точка ЕС – це точка ЕС {x, y}, представлена в її короткій формі {x, непарний / парний}. Відкриті ключі ECC є точками ЕС, тому їх також можна стиснути таким же чином.

Щоб розпакувати точку, ми можемо обчислити дві її можливі координати y за формулами:

$$y_1 = \text{mod}\sqrt{(x^3 + ax + b, p)}$$

$$y_2 = p - \text{mod}\sqrt{(x^3 + ax + b, p)}$$

Потім ми беремо **непарну** або **парну** з наведених вище координат (згідно з додатковим бітом парності в стиснутому представленні) [14].

Модульний квадратний корінь можна обчислити за допомогою алгоритму Тонеллі–Шенкса [5].

Візьмемо приклад: на еліптичній кривій $y^2 \equiv x^3 + 7\ (mod\ 17)$ точку P {10, 15} можна стиснути як C {10, непарне}. Для декомпресії ми спочатку обчислюємо дві можливі координати $y$ для $x = 10$ за допомогою наведених вище формул: $y_1 = 2$ і $y_2 = 15$. Потім ми вибираємо непарну: $y = 15$. Точкою декомпресії є {10, 15}.

```
pip install nummaster
from nummaster.basic import sqrtmod

def compress_point(point):
    """
    Compresses a point on an elliptic curve.
    Args:
        point (tuple): The (x, y) coordinates of the point.

    Returns:
        tuple: Compressed representation of the point.
    """
    return (point[0], point[1] % 2)

def uncompress_point(compressed_point, p, a, b):
    """
    Uncompresses a point on an elliptic curve.
    Args:
```

```
            compressed_point (tuple): Compressed representation of the
point.
            p (int): Prime modulus.
            a (int): Coefficient in the elliptic curve equation.
            b (int): Coefficient in the elliptic curve equation.

        Returns:
            tuple: Uncompressed (x, y) coordinates of the point.
        """
        x, is_odd = compressed_point
        y = sqrtmod(pow(x, 3, p) + a * x + b, p)

        # Check if y should be even or odd based on the compressed
representation
        if bool(is_odd) == bool(y & 1):
            return (x, y)
        return (x, p - y)

    # Define elliptic curve parameters
    p, a, b = 17, 0, 7

    # Define the original point
    point = (10, 15)
    print(f"Original point = {point}")

    # Compress the original point
    compressed_point = compress_point(point)
    print(f"Compressed point = {compressed_point}")

    # Uncompress the compressed point
    restored_point = uncompress_point(compressed_point, p, a, b)
    print(f"Restored point = {restored_point}")
```

Результат:

```
Original point = (10, 15)
Compressed point = (10, 1)
Restored point = (10, 15)
```

У криптографії ECC [14] використовуються еліптичні криві над скінченними полями, де модуль p і порядок n є дуже великими цілими числами (n зазвичай є простим числом), напр. 256-бітне число. Скінченне поле кривої має квадратну форму розміром $p \times p$, який є неймовірно великим, і всі можливі точки EC на кривій (порядок кривої n) також є дуже великим цілим числом, наприклад. 256-біт. Ми вже знаємо, що 256-бітна

крива (це означає, що p і n є 256-бітними числами) забезпечує 128-бітну надійність безпеки, а це означає, що для пошуку закритого ключа з відкритого ключа або підпису, найкращий відомий неквантовий алгоритм займає приблизно 2128 операцій. Визначена вище крива ECC secp256k1 має 128-бітну силу.

**Приклад Python із кривою "secp256k1"** [14]

Тепер давайте застосуємо наведені вище параметри домену для кривої secp256k1. Визначимо EC і розрахуємо відкритий ключ для певного закритого ключа:

```
import tinyec.ec as ec

def generate_secp256k1_curve():
    """
    Generates the secp256k1 elliptic curve.
    Returns:
        Curve: secp256k1 elliptic curve object.
    """
    name = 'secp256k1'
                                                        p               =
0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f
                                                        n               =
0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141
                                                        a               =
0x0000000000000000000000000000000000000000000000000000000000000000
                                                        b               =
0x0000000000000000000000000000000000000000000000000000000000000007
                                                        g               =
(0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798,

0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8)
    h = 1

    return ec.Curve(a, b, SubGroup(p, g, n, h), name)

secp256k1_curve = generate_secp256k1_curve()

# Define a private key in hexadecimal
privKey_hex                                                                     =
'51897b64e85c3f714bba707e867914295a1377a7463a9dae8ea6a8b914246319'
privKey = int(privKey_hex, 16)
print('privKey:', hex(privKey)[2:])

# Calculate the corresponding public key
```

```
    pubKey = secp256k1_curve.g * privKey

    # Compress the public key
    pubKeyCompressed = '0' + str(2 + pubKey.y % 2) +
str(hex(pubKey.x)[2:])
    print('pubKey:', pubKeyCompressed)
```

Наведений вище код визначає криву secp256k1 через її параметри домену та обчислює відкритий ключ за заданим закритим ключем. Це робиться шляхом множення генератора кривої G на закритий ключ. Результат правильний, як це видно з виводу програми:

```
    privKey:
51897b64e85c3f714bba707e867914295a1377a7463a9dae8ea6a8b914246319
    pubKey:
02f54ba86dc1ccb5bed0224d23f01ed87e4a443c47fc690d7797a13d41d2340e1a
```

# Висновки

Отже, дана робота була присвячена реалізації криптографічного методу еліптичних кривих для техногології блокчейн. У роботі ми дослідили головні функції криптографії і її вплив на блокчейн. Змоделювали систему Діффі-Хелмна для еліптичних кривих, проаналізувати підходи до розв'язання задачі дискретного логарифмування. Ми розібрали з чого складається блокчейн, мережа Bitcoin.

В третьому розділі ми розглянули реалізацію криптографічних еліптичних кривих над скінченним полем на мові програмування Python для різних довжин кривих, зокрема для кривої secp256k1 (кривої Bitcoin). Також ми отримали візуалізації кривих за допомогою системи Mathematica.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System / Satoshi Nakamoto – Режим доступу до ресурсу: https://bitcoin.org/bitcoin.pdf.
2. https://v-variant.com.ua/shcho-take-kryptohrafiia/
3. Мережа Bitcoin [Електронний ресурс] – Режим доступу до ресурсу: https://bitcoin.org/uk/.
4. Блокчейн [Електронний ресурс] https://uk.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA%D1%87%D0%B5%D0%B9%D0%BD
5. Tonelli–Shanks algorithm [Електронний ресурс] https://en.wikipedia.org/wiki/Tonelli%E2%80%93Shanks_algorithm
6. Poon J. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments / J. Poon, T. Dryja – Режим доступу до ресурсу: https://lightning.network/lightning-network-paper.pdf.
7. Bowe S. BIP-199 [Електронний ресурс] / S. Bowe, D. Hopwood – Режим доступу до ресурсу: https://github.com/bitcoin/bips/blob/master/bip-0199.mediawiki.
8. Smart, N.: Cryptography – An Introduction. McGraw-Hill, London (2003).
9. Partho Sutra Dhor: The Mathematics behind Blockchain.
10. N. Koblitz, A. Menezes and S. Vanstone. The state of elliptic curve cryptography. Designs Codes and Cryptography, 19, 173–193, 2000.
11. K. McCurley. The discrete logarithm problem. In Cryptology and Computational Number Theory, Proc. Symposia in Applied Maths, Volume 42, 1990.
12. A. Odlyzko. Discrete logarithms: The past and the future. Designs Codes and Cryptography, 19, 129–145, 2000.
13. Hess, F. (2005). Weil Descent Attacks. In I. Blake, G. Seroussi, & N. Smart (Eds.), Advances in Elliptic Curve Cryptography (London Mathematical Society Lecture Note Series, pp. 151-180). Cambridge: Cambridge University Press. doi:10.1017/CBO9780511546570.010

14. Svetlin Nakov, Practical cryptography for developers, 2018.