МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття ступеня вищої освіти рівня «бакалавр»

(освітньо-кваліфікаційний рівень)

на тему

«Дискретне логарифмування на еліптичних кривих»

«Discrete logarithm on an elliptic curve»

Виконав: студент денної форми навчання
напряму підготовки 123 - Комп'ютерна інженерія.

(шифр і назва напряму підготовки, спеціальності)

Гула Максим Васильович

(прізвище, ім'я, по-батькові)

Керівник _____

(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент _____

(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№___ від «___»_____ 2021 р.

Завідувач кафедри

_____ _Варбанець П.Д._

(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___

протокол № __ від «___» ____ 2021 р.

Оцінка _____/_____/_____

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ _Н.Ф.Казакова_

(підпис) (прізвище, ініціали)

Одеса - 2021

АНОТАЦІЯ

У дипломній роботі розглядається тема «Дискретне логарифмування на еліптичних кривих». Метою даної кваліфікаційної роботи є зіставлення огляду на дієві алгоритми дискретного логарифмування на еліптичній кривій, та програмна реалізація цих алгоритмів. Огляд алгоритму «Baby-step, giant-step» та p-методу Полларда, їх програмна реалізація з використанням високорівневої мови програмування Python.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

NIST – Національний інститут стандартів і технологій США;

NSA – Агенство національної безпеки США;

Unicode - стандарт кодування символів;

ЕП – електронний підпис;

DES - Data Encryption Standard, симетричний алгоритм шифрування;

AES - Advanced Encryption Standard, симетричний алгоритм блочного шифрування;

RSA - абревіатура від прізвищ Rivest, Shamir і Adleman, криптографічний алгоритм з відкритим ключем;

ВСТУП

Разом з розвитком людства стрімко розвиваються і телекомунікаційні технології, які вже сьогодні здатні об’єднати людей з різних куточків світу та забезпечити їх можливістю швидкого спілкування майже без обмежень. Захист інформації перетворюється сьогодні на одну з найактуальніших задач внаслідок надзвичайно широкого розповсюдження як власне різноманітних систем обробки інформації, так і розширення локальних та глобальних комп’ютерних мереж, якими передаються величезні об’єми інформації державного, військового, комерційного, приватного характеру, власники якої часто були б категорично проти ознайомлення з нею сторонніх осіб. Проблема набуває особливої гостроти після прийняття урядом України закону про захист персональних даних, який зобов’язує зберігати та передавати персональні дані працівників лише у захищеному вигляді в інформаційних системах. Не менш важливим завданням вважається широке впровадження інформаційних технологій у різні сфери людської діяльності в Україні: стрімке зростання обігу пластикових карток, майбутнє введення електронних паспортів та медичних карт, студентських квитків та залікових книжок; зрештою все більше державних установ та приватних підприємств переходять на електронний документообіг, який до того ж, вимагає юридичної чинності підпису фізичної або юридичної особи. Розповсюдження таких технологій також, безперечно, вимагає добре поставленого захисту інформації. Безпека сучасних інформаційних систем та технологій ґрунтується на стійкості криптографічних перетворень, які вони використовують для криптографічної обробки інформації. Криптографічна стійкість базується на складності розв’язання певних математичних задач (факторизації великого цілого числа, розв’язку дискретного логарифма тощо), для таких задач характерна субекспоненційна або експоненційна складність розв’язання на сучасних (класичних) комп’ютерах. Задача дискретного логарифмування є однією з найважливіших в області інформаційної безпеки. Проблема дискретного

логарифмування була і є актуальною, саме на припущенні про обчислювальну складність цієї задачі частіше всього базується доводження про стійкість тієї, чи іншої криптографічної системи. Слід зазначити, що з розвитком постквантової криптографії всі асиметричні криптосистеми піл загрозою. Так наприклад NIST та NSA офіційно анонсували необхідність переходу до криптографії стійкої до атак на квантовому комп'ютері.

Метою даної кваліфікаційної роботи є зіставлення огляду на дієві алгоритми дискретного логарифмування на еліптичній кривій, та програмна реалізація цих алгоритмів. Огляд алгоритму «Baby-step, giant-step» та ρ-методу Полларда, їх програмна реалізація з використанням високорівневої мови програмування Python.

ІСТОРІЯ КРИПТОГРАФІЇ

Криптографію як науку характеризують предмет, цілі та методи дослідження. Згідно з наведеними вище визначеннями, предметну область криптографії складають функції захисту інформації, реалізовані за допомогою як загальних підходів (методів), так і відповідних засобів. Ці визначення містять ряд загальних положень, серед яких можна виділити наступні:

1) криптографія - наука, що займається вивченням методів і засобів захисту інформації, зокрема, методів і засобів забезпечення конфіденційності і автентичності даних;
2) криптографія - розділ прикладної математики і, отже, використовує математичні (в першу чергу алгебраїчні) методи дослідження. Якщо звернутися до численним публікаціям в області криптографії, то можна помітити характерні риси методів дослідження, які використовуються криптографією:
   - способи забезпечення та подолання захисних функцій, які показують, що, від чого і за яких умов може бути захищене, можна описати ймовірносними алгоритмами, які характеризуються складністю і ймовірністю успіху;
   - методи і засоби захисту інформації, що вивчаються криптографією, можна охарактеризувати складністю їх подолання в рамках деякої формальної моделі (обчислювальної, інженерно-технічної, організаційної та т. д.).

Криптографія в тому чи іншому виді існувала на протязі всієї історії становлення людської цивілізації. Доказом цьому факту можуть бути знайдені зашифровані повідомлення, які використовувались в древньому Єгипті, Китаї та Індії приблизно три тисячі років до нашої ери. Багато прикладів існування закодованих повідомлень можна знайти в літературі минувших часів. Так, наприклад, Гомер в своїй «Іліаді» пише про зашифроване послання, котре передавав король Проітос, який бажав вбити молодого Беллерофонтеса. Однак

деталі багатьох стародавніх шифрувальних систем не змогли зберегтися до наших часів. Квадрат Полібія (рис. 1.1) - одна з перших таких систем, структура якої змогла не загубитися в історії людства. Вона була розроблена грецьким істориком Полібієм в другому столітті до н.е. і представляла собою квадрат 5 х 5, в кожну клітину якого була вписана буква алфавіту. У підсумку, передаючи повідомлення, кожну його букву можна було б уявити за допомогою квадрата Полібія, адже кожній букві відповідали дві цифри - номер ряду і номер стовпця. Зашифроване повідомлення складалося як раз з безлічі пар таких цифр. Однак ж головною перевагою даного методу шифрування була аж ніяк не його комплексність і надійність. Очевидною перевагою вище описаного методу кодування повідомлення була його простота в передачі повідомлень на великі відстані. Алфавіт скорочувався до виду, зручного для комунікації з допомогою, наприклад, факелів: від одного до п'яти в кожній руці (відповідно до номера рядка і стовпчика даної літери). Надалі з схожих принципів і цілей були розроблені вже відомі в сучасності азбука Морзе і семафори, вже не кажучи про ASCII і Unicode (сучасне цифрове шифрування).

	1	2	3	4	5
1	Α	Β	Γ	Δ	Ε
2	Ζ	Η	Θ	Ι	Κ
3	Λ	Μ	Ν	Ξ	Ο
4	Π	Ρ	Σ	Τ	Υ
5	Φ	Χ	Ψ	Ω	

(Рисунок 1.1 – квадрат Полібія)

Одним же з найвідоміших методів шифрування є знаменитий шифр Цезаря. Опираючись на роботи древніх римських істориків, можна зробити висновок, що сам Юлій Цезар використовував шифр, що має на увазі зсув

алфавіту на певне число - ключ даного шифру - і зіставлення буквам відкритого повідомлення букв, отриманих при зсуві, враховуючи їх новий порядковий номер в отриманому алфавіті. Звідси даний метод шифрування і отримав своє ім'я. Спочатку алфавіт зміщувався на 3 символа, таким чином A ставало D, B - E і так далі. Якщо ж буква займала одну з трьох останніх позицій, то перестановка циклічно тривала через початок алфавіту. X переходило в A, Y - в B, а Z - в C. З плином часу популярність даного методу кодування не згасла, адже навіть сьогодні він повсюдно використовується на різних інтернет ресурсах і також в популярній культурі. Наприклад, користувачі інтернету можуть активувати мод, який називається ROT13, який кодує весь контент інтернет браузера за допомогою шифру Цезаря із зсувом на 13 символів. Також, відсилання на шифр можна помітити в творі Артура Кларка «2001», де HAL - ім'я комп'ютера, є шифром Цезаря із зсувом 1 і відкритим повідомленням IBM. За простотою цього шифру криється простота в декодуванні. Для текстів англійською існує всього 25 можливих шифрів, тому для його злому потрібно перебрати лише 25 можливих варіантів тексту, поки ми не зможемо його прочитати. Більш безпечна схема зашифровуваної повідомлення має на увазі заміну кожної букви вихідного відкритого повідомлення на будь-яку іншу, але без прив'язки до певного відстані між буквами. Ключем такого шифру, в свою чергу, є таблиця відповідності між літерами. Така схема кодування повідомлення називається підстановлювальним шифром. Також, в 15-му столітті в арабській криптографічного енциклопедії вперше були згадані шифри, в яких різні закодовані символи можуть відповідати одному і тому ж символу відкритого тексту. Шифри, де кожному символу відкритого тексту відповідає єдиний символ зашифрованого називаються моноалфавітним, якщо ж більше одного - поліалфавітним. Одна з можливих тактик при зломі моноалфавітних шифрів - підрахунок частоти використання символів в зашифрованому тексті і подальше зіставлення цих частот частотам символів латинського або будь-якого іншого алфавіту. Такий метод називається частотним аналізом. При

прямому ж переборі ми стикаємося з тим, що потрібно перебрати величезну кількість можливих варіантів. Так, для моноалфавітних підстановлювальних шифрів ми можемо уявити А 26-ма варіантами, В - 25-а, С - 24-ма і так далі. У підсумку виходить 26! можливих кодувань, що майже неможливо перебрати навіть з урахуванням потужностей сучасного обладнання.

СУЧАСНА КРИПТОГРАФІЯ

Для сучасної криптографії характерне використання відкритих алгоритмів шифрування, що припускають використання обчислювальних засобів. Відомо більше десятка перевірених алгоритмів шифрування, які при використанні ключа достатньої довжини і коректної реалізації алгоритму криптографічно стійкі. Поширені алгоритми:

1) Симетричні DES, AES, Camellia, Twofish, Blowfish, IDEA, RC4 та інші.
2) Асиметричні RSA та Elgamal (Эль-Гамаль).
3) Хеш-функції MD4, MD5, MD6, SHA-1, SHA-2.

Криптографічні методи стали широко використовуватися приватними особами в електронних комерційних операціях, телекомунікаціях та багатьох інших середовищах. У багатьох країнах прийняті національні стандарти шифрування. У 2001 році в США прийнятий стандарт симетричного шифрування AES на основі алгоритму Rijndael з довжиною ключа 128, 192 і 256 біт. Алгоритм AES прийшов на зміну колишньому алгоритму DES, який тепер рекомендовано використовувати тільки в режимі Triple DES.

Симетричне шифрування - спосіб шифрування, в якому для шифрування і розшифрування застосовується один і той же криптографічний ключ. До винаходу схеми асиметричного шифрування єдиним дієвим способом було симетричне шифрування. Ключ алгоритму повинен зберігатися в таємниці обома сторонами, повинні здійснюватися заходи щодо захисту доступу до

каналу, на всьому шляху проходження криптограми. Алгоритм шифрування вибирається сторонами до початку обміну повідомленнями. Отже, головним принципом є умова, що передавач і приймач заздалегідь знають алгоритм шифрування, а також ключ до повідомлення, без яких інформація є всього лише набором символів, які не мають сенсу.

Прикладами таких алгоритмів є:

1) Проста перестановка
2) Одиночна перестановка по ключу
3) Подвійна перестановка
4) Перестановка «Магічний квадрат»

Порівняння з асиметричними криптосистемами:

Переваги	Недоліки
Швидкість.	Складність управління ключами у великій мережі.
Простота реалізації.	Складність обміну ключами. Для застосування необхідно вирішити проблему надійної передачі ключів кожному абоненту, так як потрібен секретний канал для передачі кожного ключа обом сторонам.
Менша необхідна довжина ключа для порівнянної стійкості.	
Вивченість.	

Асиметричне шифрування - система шифрування або електронного підпису (ЕП), при якій відкритий ключ передається по відкритому (тобто незахищеному, доступному для спостереження) каналу і використовується для перевірки ЕП та для шифрування повідомлення. Для генерації ЕП і для розшифровки повідомлення використовується закритий ключ. Криптографічні системи з відкритим ключем в даний час широко застосовуються в різних мережевих протоколах, зокрема, в протоколах TLS і його попереднику SSL (що лежать в основі HTTPS), в SSH.

КРИПТОГРАФІЯ НА ЕЛІПТИЧНИХ КРИВИХ

Криптосистеми на еліптичних кривих відносяться до класу криптосистем з відкритим ключем. Їх безпека, як правило, базується на труднощі рішення задачі дискретного логарифмування в групі точок еліптичної кривої над кінцевим полем. Цим і зумовлена їх висока криптостійкість в порівнянні з іншими алгоритмами. Існують стійкі криптоалгоритми на еліптичних кривих, засновані на труднощі розкладання великих цілих чисел, коли еліптична крива задається над кінцевим кільцем по складеному модулю, але вони зустрічаються досить рідко. Однак, слід зазначити, що криптостійкість є відносним поняттям, пов'язаним з поняттям найкращого відомого алгоритму злому системи.

Еліптичні криві - математичний об'єкт, який може бути визначений над будь-яким полем. У криптографії зазвичай використовуються кінцеві поля. Для точок на еліптичній кривій вводиться операція додавання, яка відіграє ту ж роль, що і операція множення в криптосистемах RSA і Ель-Гамаля. Ще однією перевагою криптосистем на еліптичних кривих є висока швидкість обробки інформації. Але і тут не все так просто. Зрозуміло, що, володіючи більш високою криптостійкістю, криптосистеми на еліптичних кривих дозволяють використовувати ключ меншої довжини.

Криптосистеми на еліптичних кривих, як, втім, і інші криптосистеми з відкритим ключем, недоцільно застосовувати для шифрування великих обсягів даних. Але зате їх можна ефективно використовувати для систем цифрового підпису та ключового обміну. З 1998 року використання еліптичних кривих для вирішення криптографічних завдань, таких як цифровий підпис, було закріплено в стандартах США ANSI X9.62 і FIPS 186-2, а в 2001 році аналогічний стандарт - ГОСТ Р34.10-2001 був прийнятий в Росії.

В Україні прийнятий стандарт цифрового підпису, заснований на еліптичних

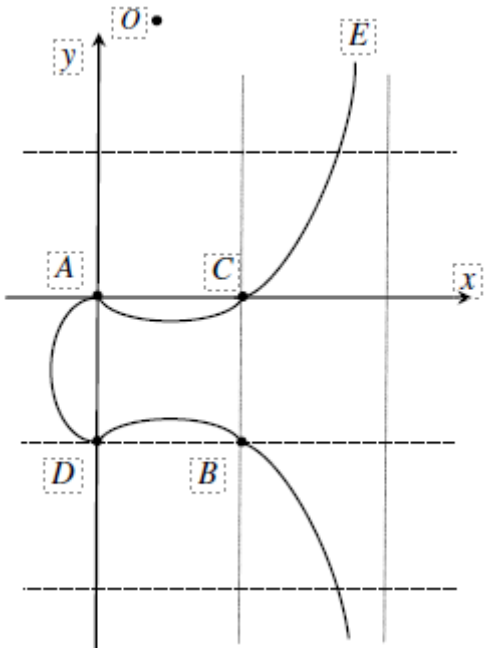
кривих ДСТУ 4145-2002. Зауважу, що безпека таких систем цифрового підпису спирається не тільки на стійкість алгоритму на еліптичних кривих, але і на стійкість використовуваної хеш-функції. Численні дослідження показали, що криптосистеми на основі еліптичних кривих перевершують інші системи з відкритим ключем по двом важливим параметрам: ступеня захищеності в розрахунку на кожен біт ключа і швидкодії при програмної і апаратної реалізації. Це пояснюється тим, що для обчислення обернених функцій на еліптичних кривих відомі тільки алгоритми з ростом трудомісткості, тоді як для звичайних систем запропоновані субекспоненціальні методи. В результаті той рівень стійкості, який досягається, скажімо, в RSA при використані1024-бітових модулів, в системах на еліптичних кривих реалізується при розмірі модуля 160 біт, що забезпечує більш просту як програмну, так і апаратну реалізацію.

Еліптичні криві (ECC - Elliptic curve cryptography) - це не еліпси. Вони так називаються просто тому, що описуються кубічними рівняннями, подібними до тих, які використовуються для обчислення кривої еліпса. У загальному випадку кубічні рівняння для еліптичних кривих мають вигляд

$$y^2 + axy + by = x^3 + cx^2 + dx + e,$$

де *a*, b, *c*, *d*, і e є дійсними числами, що задовольняють деяким простим умовам. Визначення еліптичної кривої включає також певний елемент, що позначається O , який ще називають невласним елементом (а також нескінченним елементом, або нульовим елементом). Такі рівняння називаються кубічними, або рівняннями третього порядку, оскільки в них найвищий показник ступеня дорівнює трьом.

Розглянемо еліптичну криву *E* (рис. 1.2), що відповідає рівнянню $y^2 + y = x^3 - x^2$. На цій кривій лежать тільки чотири точки, координати яких є цілими числами. Це точки *A* (0, 0), *B* (1, -1), *C* (1, 0), *D* (0, -1).

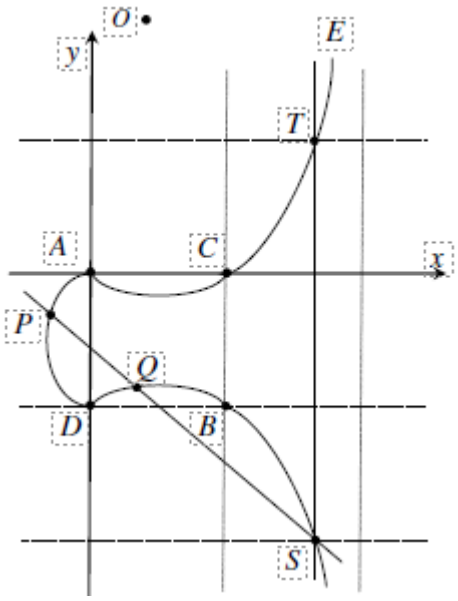


(Рисунок 1.2 – Група із п’яти точок еліптичної кривої E;
O – безкінечно віддалена точка)

Для визначення операції додавання на групі точок еліптичної кривої будемо вважати, що:

1) на площині існує нескінченно віддалена точка $O \in E$, в якій сходяться всі вертикальні прямі;
2) дотична до кривої перетинає точку дотику P два рази.

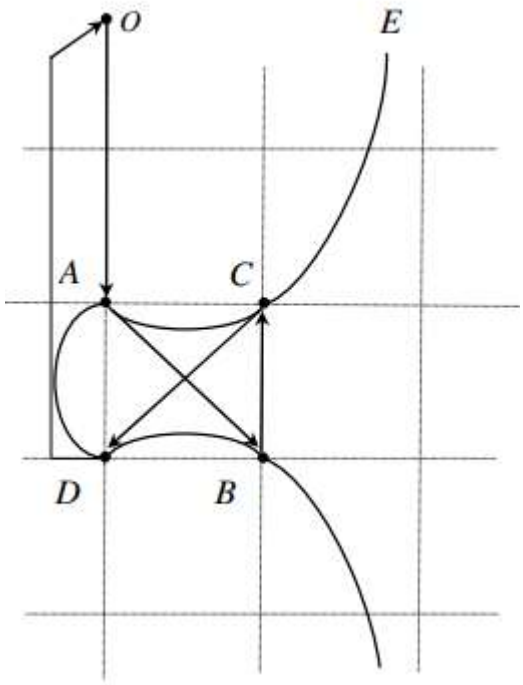
Тепер можна сформулювати правила додавання точок $P, Q \in E$ (рис. 1.3):



(Рисунок 1.3 – Додавання точок на еліптичній кривій $P + Q = T$)

1) проведемо пряму лінію через точки P і Q, знайдемо третю точку S перетину цієї прямої з кривою E;
2) проведемо через точку S вертикальну пряму до перетину з кривою E в точці T;
3) шукана сума P + Q = T.

Застосувавши ці правила до групи точок $G = \{A, B, C, D, O\}$, отримаємо (рис. 1.4):



(Рисунок 1.4 – Адитивна абелева група *{A, B, C, D, O}*

на еліптичній кривій *E*)

$$A + A = B;\ A + B = C;\ A + C = D;\ A + D = O,$$

або

$$2A = B;\ 3A = C;\ 4A = D;\ 5A = O;\ 6A = A.$$

Для будь-яких точок $P, Q \in G$ справедливо відношення $P + Q = Q + P$. Для будь-якої точки $P \in G$ справедливо $P + O = P$; інакше кажучи, точка O - адитивний одиничний елемент групи G.

У реальних криптосистемах використовується рівняння $y^2 \equiv x^3 + ax + b(mod\ p)$ де $a, b \in GF(p)$, $4a^3 + 27b^2(mod\ p) \neq 0$, $p > 3$ - просте. Група $E(GF(p))$ складається із всіх точок $(x, y); x, y \in GF(p)$, що задовольняють рівняння, і нескінченно віддаленої точки O.

Множина $E_p(a, b)$ складається з усіх точок $(x, y), x \geq 0, p > y$, які задовольняють умови рівняння $y^2 \equiv x^3 + ax + b\ (mod\ p)$, і нескінченно віддаленої точки O. Кількість точок в $Ep(a, b)$ будемо позначати $\#Ep(a, b)$. Ця величина має велике значення для криптографічних додатків еліптичних кривих.

описана наступним чином.

1) $P + O = O + P = P$.
2) Якщо $P = (x, y)$, то $P + (x, -y) = O$. Точка $(x, -y)$ є від'ємним значенням точки P і позначається $-P$. Зауважимо, що $(x, -y)$ лежить на еліптичній кривій і належить до $Ep\ (a, b)$. Наприклад, в разі $E_{23}\ (1, 1)$ для $P = (13, 7)$ маємо $-P = (13, -7)$. Але $-7 \bmod 23 \equiv 16$, таким чином, $-P = (13, 16)$.
3) Якщо $P = (x_1, y_1)$ і $Q = (x_2, y_2)$, то $P + Q = (x_3, y_3)$ визначається відповідно до правил:

$$x_3 \equiv \lambda^2 - x_1 - x_2 (mod\ p);$$

$$y_3 \equiv \lambda(x_1 - x_3) - y_1 (mod\ p),$$

де

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}, & \text{якщо } P \neq Q \\ \frac{3x_1^2 + a}{2y_1}, & \text{якщо } P = Q. \end{cases}$$

Число λ - кутовий коефіцієнт січної, проведеної через точки $P = (x_1, y_1)$ і $Q = (x_2, y_2)$. При $P = Q$ січна перетворюється в дотичну, чим і пояснюється наявність двох формул для обчислення λ.

Розглянемо основні рекомендації щодо вибору параметрів еліптичної кривої, призначеної для вирішення криптографічних завдань, а саме завдань з вибору коефіцієнтів a, b і модуля p. Фактично критерієм вибору служить неможливість здійснення певного роду атак, запропонованих для деяких класів кривих. Викладені нижче рекомендації слідують стратегії вибору випадкової кривої. Ця стратегія вважається найбільш надійним з точки зору забезпечення стійкості результуючої криптосистеми. Альтернативний підхід, що не розглянутий тут, полягає в систематичному конструюванні кривої з заданими властивостями, що зазвичай виявляється більш ефективним з обчислювальної точки зору. Для реалізації цього підходу запропоновані

спеціальні методи, але одержувані криві фактично вибираються з відносно невеликого класу і викликають підозри на наявність деяких специфічних властивостей, які можуть дозволити згодом винайти алгоритми для їх злому.

Опишу по кроках процес формування випадкової кривої.

1) Обираємо випадково просте число p. Бітова довжина числа p $t = \lfloor \log p \rfloor + 1$ повинна бути такою, щоб унеможливити використання методів знаходження логарифмів на кривій, що мають трудомісткість T ($2^{t/2}$). Величина t = 128 біт (чотири машинних слова на 32-бітових комп'ютерах) сьогодні недостатня, так як є повідомлення про злом відповідних кривих. Інша точка зору заснована на тому, що шифр на еліптичній кривій повинен бути не менш стійким, ніж блоковий шифр AES (Advanced Encryption Standard). Вважається, що стійкість AES забезпечується повною довжиною його ключа, яка становить 128, 196 або 256 біт. Так як стійкість шифру на еліптичній кривій визначається величиною t / 2, довжина модулів еліптичних кривих повинна становити відповідно 256, 392 і 512 біт.
2) Обираємо випадкові числа a, і b такі, що $a, b \ (mod\ p) \neq 0$ і $4a^3 + 27b^2 (mod\ p) \neq 0$. Звернемо увагу на те, що при обчисленні композиції точок параметр b ніде не фігурує. Тому для підвищення ефективності розрахунків, іноді, рекомендують випадково вибирати тільки b, а a приймати рівним невеликому цілому числу. Так, стандарт США FIPS 186-2 передбачає використання кривих з параметром $a = -3$, що дещо спрощує обчислення.
3) Визначаємо число точок на кривій $n = \#Ep\ (a, b)$ (це самий трудомісткий етап описуваного процесу). Важливо, щоб n мало великий простий дільник q, а найкраще саме було простим числом, $n = q$. Якщо n розкладається на малі множники, то в $Ep\ (a, b)$ існує багато малих підмножин зі своїми генераторами і алгоритм

Поліга - Хеллмана швидко обчислює логарифм на кривій через логарифми в цих малих підмножинах. Якщо пошук кривої з $n = q$ займає надто багато часу, то можна допустити $n = h\ q$, де h - невелике число. Ще раз слід підкреслити, що стійкість криптосистеми на еліптичній кривій визначається не модулем p, а числом елементів q в підмножині точок кривої. Але якщо множник h - невелике число, то q є величиною того ж порядку, що і p. Якщо n не відповідає вимогам, то необхідно повернутися до кроку 2.

4) Перевіряємо, чи виконуються нерівності $(p^k - 1) \bmod q \neq 0$ для всіх k, $0 < k < 32$. Якщо ні, то повертаємося до кроку 2. Ця перевірка запобігає можливості MOV-атаки (названої по прізвищах її авторів Menezes, Okamoto, Vanstone), а також виключає з розгляду так звані суперсингулярні криві і криві з $\#Ep(a, b) = p - 1$. Метод MOV і зазначені особливі типи кривих дозволяють звести задачу обчислення логарифма на кривій до більш простих завдань.
5) Перевіряємо, чи виконується нерівність q ≠ p. Якщо ні, то повертаємось до кроку 2. Справа в тому, що для кривих з $q = p$, які називаються аномальними, існують ефективні методи обчислення логарифмів.
6) На даному етапі крива, яка підходить для криптографічних додатків отримана. Ми маємо параметри p, a, b, кількість точок n і розмір підмножини точок q. Зазвичай ще потрібно знайти точку G - генератор цієї підмножини. Якщо $q = n$, то будь-яка точка (крім O) є генератором. Якщо $q < n$, то обираємо випадкові точки G', поки не отримаємо $G = [n / q]\ G' \neq O$. Щоб отримати випадкову точку на кривій, беремо випадкове число $x < p$, обчислюємо $e \equiv (x^3 + ax + b) \bmod p$ і намагаємося витягти квадратний корінь $y = \sqrt{e} \bmod p$. Якщо корінь існує, то отримуємо точку (x, y), в іншому випадку пробуємо інше число x.

Завдання, яке вирішує криптоаналитик при використанні криптосистеми на базі еліптичних кривих, називається завданням дискретного логарифмування на еліптичній кривій і формулюється наступним чином. Дано точки P і Q на еліптичній кривій порядку n, де n - число точок на кривій. Необхідно знайти єдину точку x, таку, що $P = xQ$. Про ефективні алгоритми дискретного логарифмування мова піде далі.

АЛГОРИТМИ ДИСКРЕТНОГО ЛОГАРИФМУВАННЯ НА ЕЛІПТИЧНИХ КРИВИХ

«Baby-step, giant-step», інша назва «Дитячий крок, крок гіганта» - це алгоритм «зустрічі посередині». На відміну від атаки перебором (при якій доведеться обчислювати всі точки xP для кожного x, поки ми не знайдемо Q), можна обчислювати «кілька» значень для bP і «кілька» значень для Q - amP, поки ми не знайдемо відповідність. Задачу дискретного логарифмування $P = xQ$ можна переписати наступним чином:

$$\begin{aligned} Q &= xP \\ Q &= (am + b)P \\ Q &= amP + bP \\ Q - amP &= bP \end{aligned}$$

Алгоритм працює наступним чином:

1. Обчислюємо $m = |\sqrt{n}|$;
2. Для кожного b із $0, \ldots, m$ обчислюємо bP і зберігаємо результати до хеш-таблиці;
3. Для кожного a із $0, \ldots, m$:
    1) Обчислюємо amP;
    2) Обчислюємо $Q - amP$;

3) Перевіряємо хеш-таблицю і шукаємо точку bP таку, що $Q - amP = bP$;
4) Якщо така точка існує, то ми знайшли $x = am + b$;
5) В іншому випадку переходимо до кроку 3.

Спочатку ми обчислюємо точки bP з невеликим інкрементом («дитячими кроками» «baby step») для коефіцієнта b ($1P,2P,3P$, ...). У другій частині алгоритму ми обчислюємо точки amP з великим інкрементом («велетенські кроками» «giant step») для am($1mP,2mP,3mP$, ..., де m - велике число).

Візьмем стандартизованну криву: prime192v1 (вона ж secp192r1, ansiX9p192r1). Ця крива має порядок n = 0xffffffff ffffffff ffffffff 99def836 146bc9b1 b4d22831. Квадратний корінь з n - це приблизно $7{,}922816251426434 \cdot 10^{28}$ (майже вісімдесят октілліонов). Уявімо, що ми зберігаємо $|\sqrt{n}|$ точок в хеш-таблиці. Припустимо, що кожна точка займає рівно 32 байта: для хеш-таблиці буде потрібно приблизно $2{,}5 \cdot 10^{30}$ байт пам'яті. Сучасна загальна ємність накопичувачів усього світу має порядок зеттабайта (10^{21} байт). Це майже на десять порядків менше, ніж обсяг пам'яті, необхідний нашій хеш-таблиці! Навіть якби точки займали по 1 байт кожна, ми все одно не змогли б зберігати їх всі. Приклад роботи програми (Рис. 1.5)

```
Крива: y^2 = (x^3 + 1x - 1) mod 10177
Порядок кривої: 10331
p = (0x1, 0x1)
q = (0x22e0, 0xe09)
8489 * p = q
log(p, q) = 8489
Виконано 185 кроків
>>>
=======================================
Крива: y^2 = (x^3 + 1x - 1) mod 10177
Порядок кривої: 10331 ;
p = (0x1, 0x1) ;
q = (0xed9, 0xff4) ;
1722 * p = q
log(p, q) = 1722 ;
Виконано: 118 кроків.
>>>
=======================================
Крива: y^2 = (x^3 + 1x - 1) mod 10177
Порядок кривої: 10331 ;
p = (0x1, 0x1) ;
q = (0x1b91, 0x20c2) ;
9302 * p = q
log(p, q) = 9302 ;
Виконано: 193 кроків.
```

(Рис. 1.5 – Приклад роботи алгорима «Baby-step, giant-step»)

Програмну реалізацію алгоритма можна переглянути в Додатку А.

ρ -метод Полларда - це ще один алгоритм обчислення дискретних логарифмів. В ρ-методі Полларда будемо вирішувати трохи іншу задачу: знайти для заданих P і Q цілі a, b, A, B такі, що $aP + bQ = AP + BQ$. Знайшовши чотири цілих числа, ми зможемо використовувати рівняння для обчислення $Q = xP$:

$$\begin{aligned} aP + bQ &= AP + BQ \\ aP + bxP &= AP + BxP \\ (a + bx)P &= (A + Bx)P \\ (a - A)P &= (B - b)xP \end{aligned}$$

Наша підгрупа циклічна і має порядок n, тобто коефіцієнти, використовувані при множенні точок, беруться по модулю n:

$$\begin{aligned} a - A &\equiv (B - b)x \pmod{n} \\ x &= (a - A)(B - b)^{-1} \bmod n \end{aligned}$$

Принцип роботи ρ Полларда простий: ми визначаємо псевдовипадкову послідовність пар *(a, b)*. Цю послідовність пар можна використовувати для генерування послідовності точок $aP + bQ$. Оскільки *P* і *Q* є елементами однієї циклічної підгрупи, послідовність точок теж циклічна. Це означає, що якщо ми обійдемо нашу псевдовипадкову послідовність пар *(a, b)*, то рано чи пізно виявимо цикл. Тобто: ми знайдемо пару *(a, b)* і іншу окрему пару *(A, B)*, такі, що $aP + bQ = AP + BQ$. Ті ж точки, окремі пари: ми зможемо застосувати наведене вище рівняння для знаходження логарифма. Нам потрібно знайти цей самий цикл найбільш ефективним чином. Для цього існує алгоритм черепахи і зайця, також відомий як алгоритм знаходження цикла Флойда. В чому полягає суть алгоритма, наприклад у нас є крива $y^2 \equiv x^3 + 2x + 3\ (mod 97)$ і точки *P(3,6)* та *Q(80,87)*. Точки належать циклічній підгрупі з порядком 5. Ми обходимо послідовність пар з різними швидкостями, поки не знаходимо дві різні пари *(a, b)* і *(A, B)*, що дають одну точку. В цьому випадку ми знайшли пари (3,3) і (2,0), що дозволяє нам обчислити логарифм як $x = (3 - 2)(0 - 3)^{-1} mod\ 5 = 3$. І насправді, вийшло $Q = 3P$. Ми беремо двох тваринок, черепаху і зайця, і змушуємо обходити послідовність зліва направо. Черепаха повільна і зчитує кожну точку, одну за одною; заєць швидкий і пропускає точку на кожному кроці. Через якийсь час черепаха і заєць знайдуть одну точку, але з різними парами коефіцієнтів. Або, якщо висловити це рівняннями, черепаха знайде пару *(a,b)*, а заєць – пару *(A, B)*, такі, що $aP + bQ = AP + BQ$. Цікавий факт у 1998 році Certicom почала змагання по обчисленню дискретних логарифмів на еліптичних кривих з бітовою довжиною від 109 до 369 бит. На сьогоднішній день успішно зламані тільки криві довжиною 109 біт. Остання успішна спроба була здійснена в 2004 році. Нагорода була вручена 8 квітня 2004 року близько 2600 людей, яких представляв Кріс Моніко. Вони теж використовували різновид

розпаралеленого p-метода Полларда, обчислення зайняли 17 місяців календарного часу. Приклад роботи *p*-методу Полларда (Рис. 1.6).

```
Curve: y^2 = (x^3 + 1x - 1) mod 10177
Curve order: 10331
p = (0x1, 0x1)
q = (0x1c33, 0xe94)
7567 * p = q
log(p, q) = 7567
Took 110 steps
>>>
========================================
Крива: y^2 = (x^3 + 1x - 1) mod 10177
Порядок кривої: 10331
p = (0x1, 0x1)
q = (0x13b4, 0xc41)
8819 * p = q
log(p, q) = 8819
Виконано 144 кроків
>>>
========================================
Крива: y^2 = (x^3 + 1x - 1) mod 10177
Порядок кривої: 10331
p = (0x1, 0x1)
q = (0x2383, 0xc03)
9208 * p = q
log(p, q) = 9208
Виконано 63 кроків
```

(Рис. 1.6 - Приклад роботи *p*-методу Полларда)

Програмну реалізацію метода можна переглянути в Додатку А.

ВИСНОВОК

В процесі написання кваліфікаційної роботи було реалізовано два алгоритма для дискретного логарифмування на еліптичних кривих, а саме «Baby-step, giant-step», а також p-метод Полларда. Був проведений огляд на стан криптографії в даний час, її історія, переваги та недоліки різних криптосистем. Робота може бути використана як методичний посібник в темі еліптичної криптографії. В подальшому можна покращити роботу запрограмованих методів, наприклад розпаралелити p-метод Полларда.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. А.Г. Ростовцев, Е.Б. Маховенко. «Теоретическая криптография» / М.: Изд-во «Профессионал» 2000-2003гг. г. Санкт-Петербург – с. 294;
2. Шнайер Брюс. «Прикладная криптография. Протоколы, алгоритмы и исходный код на С» / М.: Изд-во «Вильямс» 2016г. г. Москва;
3. Фергюсон Н., Шнайер Б., «Практическая криптография» /М.: Изд-во «Вильямс» 2005г. 424с.;
4. Болотов А. А.,«Элементарное введение в эллиптическую криптографию. Протоколы криптографии на эллиптических кривых» /М.: Ком.Книга 2006. -280с.;

ДОДАТОК А

ПРОГРАМНА РЕАЛІЗАЦІЯ

Класс для роботи з еліптичними кривими:

```
class EllipticCurve:
    def __init__(self, p, a, b, g, n):
        self.p = p
        self.a = a
        self.b = b
        self.g = g
        self.n = n
        assert pow(2, p - 1, p) == 1
        assert (4 * a * a * a + 27 * b * b) % p != 0
        assert self.is_on_curve(g)
        assert self.mult(n, g) is None
    def is_on_curve(self, point):
        """Перевіряємо, чи лежить дана точка на еліптичній
кривій."""
        if point is None:
            return True
        x, y = point
        return (y * y - x * x * x - self.a * x - self.b) %
self.p == 0
    def add(self, point1, point2):
        """Повертає результат точки1 + точки2 згідно із груповим
законом."""
        assert self.is_on_curve(point1)
        assert self.is_on_curve(point2)
        if point1 is None:
            return point2
        if point2 is None:
            return point1
        x1, y1 = point1
        x2, y2 = point2
        if x1 == x2 and y1 != y2:
            return None
        if x1 == x2:
            m = (3 * x1 * x1 + self.a) * inverse_mod(2 * y1,
self.p)
        else:
            m = (y1 - y2) * inverse_mod(x1 - x2, self.p)
        x3 = m * m - x1 - x2
        y3 = y1 + m * (x3 - x1)
        result = (x3 % self.p,
                  -y3 % self.p)
        assert self.is_on_curve(result)
        return result
```

```
    def double(self, point):
        """Повертає 2 * точку."""
        return self.add(point, point)

    def neg(self, point):
        """Повертає -точка."""
        if point is None:
            return None

        x, y = point
        result = x, -y % self.p

        assert self.is_on_curve(result)

        return result

    def mult(self, n, point):
        """Повертає n * точку, обчислену за допомогою алгоритму
double та add."""
        if n % self.n == 0 or point is None:
            return None

        if n < 0:
            return self.neg(self.mult(-n, point))
        result = None
        addend = point
        while n:
            if n & 1:
                result = self.add(result, addend)
            addend = self.double(addend)
            n >>= 1
        return result
    def __str__(self):
        a = abs(self.a)
        b = abs(self.b)
        a_sign = '-' if self.a < 0 else '+'
        b_sign = '-' if self.b < 0 else '+'

        return 'y^2 = (x^3 {} {}x {} {}) mod {}'.format(
            a_sign, a, b_sign, b, self.p)
def inverse_mod(n, p):
    """Повертає обернену до n за модулем p.
     Ця функція повертає єдине ціле число x таке, що (x * n)% p
== 1.
     n має бути ненульовим, а p - простим.
    """
    if n == 0:
        raise ZeroDivisionError('ділення на нуль')
    if n < 0:
        return p - inverse_mod(-n, p)
```

```
    s, old_s = 0, 1
    t, old_t = 1, 0
    r, old_r = p, n

    while r != 0:
        quotient = old_r // r
        old_r, r = r, old_r - quotient * r
        old_s, s = s, old_s - quotient * s
        old_t, t = t, old_s - quotient * t

    gcd, x, y = old_r, old_s, old_t

    assert gcd == 1
    assert (n * x) % p == 1

    return x % p


tinycurve = EllipticCurve(
    p=10177,
    a=1,
    b=-1,
    g=(1, 1),
    n=10331,
)
```

Реалізація алгоритму «Baby-step, giant-step»:

```
import math
import random

from common import tinycurve as curve


def log(p, q):
    assert curve.is_on_curve(p)
    assert curve.is_on_curve(q)

    sqrt_n = int(math.sqrt(curve.n)) + 1

    # Обчислюємо кроки дитини та зберігаємо їх у хеш-таблицю.
    r = None
    precomputed = {None: 0}

    for a in range(1, sqrt_n):
        r = curve.add(r, p)
        precomputed[r] = a

    # Тепер обчислюємо гігантські кроки та перевіряємо хеш-
таблицю на наявність
    # відповідностей.
    r = q
    s = curve.mult(sqrt_n, curve.neg(p))
```

```
    for b in range(sqrt_n):
        try:
            a = precomputed[r]
        except KeyError:
            pass
        else:
            steps = sqrt_n + b
            logarithm = a + sqrt_n * b
            return logarithm, steps

        r = curve.add(r, s)

    raise AssertionError('Логарифм не знайдено.')


def main():
    x = random.randrange(1, curve.n)
    p = curve.g
    q = curve.mult(x, p)

    print('Крива: {}'.format(curve))
    print('Порядок кривої: {}'.format(curve.n), ';')
    print('p = (0x{:x}, 0x{:x})'.format(*p), ';')
    print('q = (0x{:x}, 0x{:x})'.format(*q), ';')
    print(x, '* p = q')

    y, steps = log(p, q)
    print('log(p, q) =', y,';')
    print('Виконано:', steps, 'кроків.')

    assert x == y


if __name__ == '__main__':
    main()
```

Реалізація p-метода Полларда:

```
import random

from common import inverse_mod, tinycurve as curve


class PollardRhoSequence:

    def __init__(self, point1, point2):
        self.point1 = point1
        self.point2 = point2

        self.add_a1 = random.randrange(1, curve.n)
        self.add_b1 = random.randrange(1, curve.n)
```

```
        self.add_x1 = curve.add(
            curve.mult(self.add_a1, point1),
            curve.mult(self.add_b1, point2),
        )

        self.add_a2 = random.randrange(1, curve.n)
        self.add_b2 = random.randrange(1, curve.n)
        self.add_x2 = curve.add(
            curve.mult(self.add_a2, point1),
            curve.mult(self.add_b2, point2),
        )

    def __iter__(self):
        partition_size = curve.p // 3 + 1

        x = None
        a = 0
        b = 0

        while True:
            if x is None:
                i = 0
            else:
                i = x[0] // partition_size

            if i == 0:
                #x - або точка на нескінченності (Немає), або
знаходиться в першій
                # третині площини (x [0] <= curve.p / 3).
                a += self.add_a1
                b += self.add_b1
                x = curve.add(x, self.add_x1)
            elif i == 1:
                # x знаходиться у другій третині площини
                # (curve.p / 3 < x[0] <= curve.p * 2 / 3).
                a *= 2
                b *= 2
                x = curve.double(x)
            elif i == 2:
                # x знаходиться в останній третині площини (x
[0]> curve.p * 2/3).
                a += self.add_a2
                b += self.add_b2
                x = curve.add(x, self.add_x2)
            else:
                raise AssertionError(i)

            a = a % curve.n
            b = b % curve.n

            yield x, a, b
```

```
def log(p, q, counter=None):
    assert curve.is_on_curve(p)
    assert curve.is_on_curve(q)

    # Rho Полларда іноді може зазнати невдачі: він може знайти
a1 == a2 та b1 == b2,
     #, що веде до ділення на нульову помилку. Тому що
PollardRhoSequence використовує
     # випадкові коефіцієнти, ми маємо більше шансів знайти
логарифм
     # якщо ми спробуємо ще раз, не впливаючи на асимптотичну
складність часу.
     # Ми намагаємось щонайбільше три рази, перш ніж здаватися.
    for i in range(3):
        sequence = PollardRhoSequence(p, q)

        tortoise = iter(sequence)
        hare = iter(sequence)

        # Діапазон від 0 до curve. N - 1, але насправді алгоритм
буде
         # зупинятись набагато раніше (або знаходження
логарифму, або невдача за допомогою
         # ділення на нуль).
        for j in range(curve.n):
            x1, a1, b1 = next(tortoise)

            x2, a2, b2 = next(hare)
            x2, a2, b2 = next(hare)

            if x1 == x2:
                if b1 == b2:
                    # Це призвело б до ділення на нуль.
Спробуйте
                     # ще одна випадкову послідовність.
                    break

                x = (a1 - a2) * inverse_mod(b2 - b1, curve.n)
                logarithm = x % curve.n
                steps = i * curve.n + j + 1
                return logarithm, steps

    raise AssertionError('логарифм не знайдено')


def main():
    x = random.randrange(1, curve.n)
    p = curve.g
    q = curve.mult(x, p)

    print('Крива: {}'.format(curve))
```

```
    print('Порядок кривої: {}'.format(curve.n))
    print('p = (0x{:x}, 0x{:x})'.format(*p))
    print('q = (0x{:x}, 0x{:x})'.format(*q))
    print(x, '* p = q')

    y, steps = log(p, q)
    print('log(p, q) =', y)
    print('Виконано', steps, 'кроків')

    assert x == y


if __name__ == '__main__':
    main()
```