Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(освітній рівень)

на тему: "Дослідження стійкості криптосистем побудованих на складності знаходження дискретного логарифму"

Виконав: студент (ка) IV курсу, групи СБ-42

Спеціальності:
125 «Кібербезпека»
(шифр і назва напряму підготовки, спеціальності)
Шевчук Уляна Любомирівна
підпис (прізвище та ініціали)

Керівник Загородна Н. В.
підпис (прізвище та ініціали)

Нормоконтроль Дроздова Т. В.
підпис (прізвище та ініціали)

Завідувач кафедри Загородна Н.В.
підпис (прізвище та ініціали)

Рецензент
підпис (прізвище та ініціали)

м. Тернопіль – 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Загородна Н.В.
(підпис) (прізвище та ініціали)
«__» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 125 Кібербезпека
(шифр і назва спеціальності)

Студенту Шевчук Уляні Любомирівні
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження стійкості криптосистем побудованих на складності знаходження дискретного логарифму

Керівник роботи Загородна Наталя Володимирівна, к.т.н., доцент
Завідувач кафедри КБ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 02 » 05 2025 року № 4/7-361

2. Термін подання студентом завершеної роботи 15.06.2025

3. Вихідні дані до роботи Літературні джерела, в яких описано криптографічні системи, безпека яких ґрунтується на складності задачі дискретного логарифму

4. Зміст роботи (перелік питань, які потрібно розробити)
Теоретичні основи дискретного логарифму
Аналіз атак на криптографічні системи які побудовані на задачі дискретного логарифму
Дослідження та реалізація атаки Поліг-Хеллмана
Безпека життєдіяльності, основи охорони праці
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Дослідження стійкості криптосистем побудованих на складності знаходження дискретного логарифму
Мета
Математичні основи
Проблема дискретного логарифму
Атака на DLP
Рекомендації щодо підвищення стійкості криптографічних систем
Реалізація Поліг-Хеллмана
Програмна реалізація
Результати тестування
Висновок

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Мариненко С. Ю., к.т.н. доцент кафедри МТ		

7. Дата видачі завдання 29.01.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	29.01 - 02.02	Виконано
2.	Підбір джерел в галузі дослідження	29.01 - 05.02	Виконано
3.	Опрацювання джерел	06.02 - 20.02	Виконано
4.	Аналіз атак та їхньої реалізації	22.02 - 12.03	Виконано
5.	Розробка програмного коду	15.03 - 25.03	Виконано
6.	Тестування роботи програми	25.02 - 10.04	Виконано
7.	Оформлення розділу "Теоретичні основи дискретного логарифму"	10.02 - 05.03	Виконано
8.	Оформлення розділу " Аналіз атак на криптографічні системи, побудовані на задачі дискретного логарифму"	26.03 - 04.05	Виконано
9.	Оформлення розділу "Дослідження та реалізація атаки Pohlig-Hellman"	12.04 - 20.04	Виконано
10.	Виконання завдання до підрозділу «Безпека життєдіяльності, основи охорони праці»	25.04 - 10.05	Виконано
11.	Оформлення кваліфікаційної роботи	23.05 - 08.06	Виконано
12.	Нормоконтроль	10.06 - 15.06	Виконано
13.	Перевірка на плагіат	20.06 - 22.06	Виконано
14.	Попередній захист кваліфікаційної роботи	14.06 - 15.06	Виконано
15.	Захист кваліфікаційної роботи	27.06.2025	

Студент _____ Шевчук У. Л.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Загородна Н. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дослідження стійкості криптосистем побудованих на складності знаходження дискретного логарифму // Кваліфікаційна робота ОР "Бакалавр" // Шевчук Уляна Любомирівна // // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра кібербезпеки, група СБ-42 // Тернопіль, 2025 // С. 57, рис. - 12, табл. - 4, ліст. - 7, додат. - 1.

КЛЮЧОВІ СЛОВА: дискретний логарифм, криптосистема, Поліг-Хеллман, атака

Кваліфікаційна робота присвячена аналізу задачі дискретного логарифму та атак на криптосистеми, що ґрунтуються на цій проблемі.

Для кращого розуміння задачі, описано математичні основи, а саме групи, їхні типи і властивості. Проаналізовано проблему обчислення дискретного логарифму, як і в класичних мультиплікативних групах, так і в групах точок еліптичних кривих.

У роботі розглянуто такі атаки, як: метод повного перебору, метод Шенкса, алгоритм Полларда "По", алгоритм індексного числення, алгоритм Поліг-Хеллмана та інші. Оцінено переваги і недоліки кожного з методів. Також було проаналізовано специфіку атак на задачу дискретного логарифму на еліптичних кривих. Надано рекомендації щодо підвищення стійкості криптографічних систем до зазначених атак.

Розроблено математичну та програмну реалізації методу Поліг-Хеллмана, який дозволяє ефективно розв'язувати задачу дискретного логарифму. Проведено тестування цієї реалізації та досліджено ефективність. Отримані результати дозволяють краще зрозуміти принцип дії атаки та її обмеження.

ABSTRACT

Research on the resistance of cryptosystems built on the complexity of finding the discrete logarithm // Thesis of educational level "Bachelor" // Uliana Shevchuk // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Cybersecurity, Group SB-42 // Ternopil, 2025 // 57 pages, figures - 12, tables - 4, listings – 7, appendices – 1.

KEYWORDS: Discrete logarithm, Cryptosystem, Pohlig-Hellman, Attack

The qualification thesis is devoted to the analysis of the discrete logarithm problem and of attacks on cryptosystems that are based on this problem.

For a better understanding of the problem, the mathematical foundations are described, namely groups, their types and properties. The problem of computing the discrete logarithm has been analysed both in classical multiplicative groups and in groups of points of elliptic curves.

The thesis considers such attacks as: the brute-force method, the Shanks method, Pollard’s "Rho" algorithm, the Index Calculus algorithm, the Pohlig-Hellman algorithm and others. The advantages and disadvantages of each method have been evaluated. The specifics of attacks on the discrete logarithm problem on elliptic curves have also been analysed. Recommendations are provided for increasing the resistance of cryptographic systems to the specified attacks.

Mathematical and software implementations of the Pohlig-Hellman method have been developed, which allow the discrete logarithm problem to be solved efficiently. This implementation has been tested and its effectiveness studied. The obtained results make it possible to better understand the principle of operation of the attack and its limitations.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CRT	-	Chinese Remainder Theorem
DLP	-	Discrete Logarithm Problem
DSA	-	Digital Signature Algorithm
ECDH	-	Elliptic Curve Diffie-Hellman
ECDLP	-	Elliptic Curve Discrete Logarithm Problem
IoT	-	Internet of Things
MOV	-	Menezes-Okamoto-Vanstone
NIST	-	National Institute of Standards and Technology

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ДИСКРЕТНОГО ЛОГАРИФМУ

1.1 Алгоритми, побудовані на задачі дискретного логарифму та їх історія розвитку

Із розвитком інформаційних технологій, попит на захищеність даних значно підвищився, особливо в умовах постійного зростання загроз в інформаційному просторі. Криптографія є фундаментом для сучасної безпеки в цифровому середовищі, забезпечуючи конфіденційність та цілісність інформації. Одним з основних напрямів є криптографія з відкритим ключем, яка ґрунтується на складності різних математичних задач. Однією з важливих серед них є проблема розв’язання дискретного логарифму (англ. Discrete Logarithm Problem, DLP), яка лежить в основі багатьох сучасних криптосистем. DLP полягає в знаходженні такого показника степеню, при якому заданий елемент групи дорівнює результату піднесення генератора до цього степеня за модулем. Дослідження стійкості криптосистем, побудованих на DLP, дозволяє зрозуміти, які параметри забезпечують максимальну безпеку та які алгоритмічні підходи можуть загрожувати цій безпеці.

Виникнення DLP бере свій початок ще з 1976 року, коли Уітфілд Діффі та Мартін Хеллман запропонували перший у світі протокол обміну ключами з відкритим ключем протокол Diffie-Hellman. Простий у реалізації, доведена математична стійкість, але відсутність автентифікації та можливість атаки типу "людина посередині".

Недовго після цього, з’явився ElGamal, який був розроблений Тахером Ель-Гамалем у 1985 році, що ґрунтується на ідеї протоколу Diffie-Hellman. Цей алгоритм забезпечує повноцінну систему шифрування. ElGamal також працює на основі дискретного логарифму у скінченних циклічних групах, а також може бути адаптований до еліптичних кривих [1].

Наступним не менш важливим прикладом у розвитку алгоритмів, що побудовані на DLP є цифровий підпис (англ. Digital Signature Algorithm, DSA), який розроблений у 1991 році Національним інститутом стандартів і технологій

США (англ. National Institute of Standards and Technology, NIST). DSA є стандартом цифрового підпису в багатьох державах і установах. Цифровий підпис ґрунтується на DLP і показує високу ефективність, але для безпеки критично важливо використовувати надійний генератор випадкових чисел, інакше можливе розкриття секретного ключа.

Згодом концепція дискретного логарифму була перенесена в інші алгебраїчні структури, а саме в групи точок еліптичних кривих над скінченними полями. Це стало першопочатком еліптичної криптографії, яка дозволяє зберігати високу безпеку з меншими обчислювальними ресурсами, однак реалізація систем стала набагато складнішою і важливо правильно підібрати криву.

Серед менш відомих, але не менш цікавих криптосистем, що ґрунтуються на DLP, є Schnorr Signature Scheme, який був описаний Клаусом Шнорром у 1989 році, а запатентовано до лютого 2008 року [2]. Завдяки ефективності та короткій довжині, цей алгоритм стає все більш популярним, особливо у сфері криптовалют.

Отже, криптосистеми, що базуються на DLP, на сьогодні є одними з ключових елементів безпеки. Водночас, з розвитком квантових обчислень, розпочались активні дослідження в постквантовій криптографії, але DLP-алгоритми поки що залишаються основою для безпеки більшості класичних криптографічних систем.

1.2 Математичні основи

Однією з ключових передумов до розуміння криптосистем, заснованих на складності задачі дискретного логарифму, є базові знання з абстрактної алгебри. Саме групові структури лежать в основі побудови математичних моделей, що застосовуються в сучасній криптографії з відкритим ключем.

Група – множина елементів G разом з бінарною операцією (додавання або множення), яка задовільняє наступні властивості:

• Операція замкнена: для будь-яких $a, b \in G$ результат операції $a \circ b$ також належить G. Наприклад: якщо додати два цілих числа, то результатом теж буде ціле число.

• Операція асоціативна: для всіх $a, b, c \in G$ – $a \circ (b \circ c) = (a \circ b) \circ c$. Наприклад: при множенні неважливо в якому порядку множити.

• Існує нейтральний елемент: є такий елемент $1 \in G$, що для всіх $a \in G$ виконується $a \circ 1 = 1 \circ a = a$. Наприклад: при додаванні нейтральним елементом є 0, бо $a + 0 = a$.

• Кожен елемент має обернений: для кожного $a \in G$ існує такий $a^{-1} \in G$, що $a \circ a^{-1} = a^{-1} \circ a = 1$. Наприклад: для множення $2 \circ \frac{1}{2} = \frac{1}{2} \circ 2 = 1$.

Якщо додатково $a \circ b = b \circ a$ для всіх $a, b \in G$, тобто, що порядок немає значення, то така група називається комутативною або абелевою [3].

Групи поділяють на циклічні і нециклічні, а також на скінченні і нескінченні.

Група є скінченною, якщо вона має скінченну кількість елементів. Позначають кількість елементів (потужність) у групі як $|G|$.

Наприклад: $Z_n, +$ – додавання за модулем n, де:

• Елементи: $\{0, 1, 2, 3, n - 1\}$.

• Операція: додавання по модулю n.

• Нейтральний елемент: 0.

• Кожен елемент має обернений елемент (наприклад, для 3 в Z_5, оберненим елементом є 2).

Група є циклічною, якщо існує елемент α, порядок якого дорівнює кількості елементів у групі: $\mathrm{ord}(\alpha) = |G|$. Такий елемент називається примітивним елементом або генератором. Генератор – число, яке при піднесенні до степенів утворює (породжує) всі елементи групи без повторень. Тобто, якщо в групі є елемент, який може згенерувати всі інші, то група циклічна. Циклічні групи можуть бути як скінченними, так і нескінченними. Важливим є саме існування генератора [4]. Наприклад:

• Z_n – скінченна циклічна група з додаванням по модулю n;

•Z – нескінченна циклічна група з додаванням, бо всі цілі числа можна отримати як суми або різниці одиниці.

Наприклад, ми хочемо перевірити чи $\alpha = 2$ є генератором в групі елементів $Z_{11}^* = \{1,2,3,4,5,6,7,8,9,10\}$.

Таблиця 1.1 – Перевірка генератора

Степінь	Обчислення	Результат
2^1	$2\ mod\ 11$	2
2^2	$4\ mod\ 11$	4
2^3	$8\ mod\ 11$	8
2^4	$16\ mod\ 11$	5
2^5	$32\ mod\ 11$	10
2^6	$20\ mod\ 11$	9
2^7	$18\ mod\ 11$	7
2^8	$14\ mod\ 11$	3
2^9	$6\ mod\ 11$	6
2^{10}	$2^9 * 2 = 6 * 2 = 12\ mod\ 11$	1

У результаті, ми отримали 10 чисел із групи Z_{11}^*. Ми також бачимо, що порядок у якому вони з'являються є абсолютно випадковим. Тобто зв'язок між степенем i та елементами групи є випадковим. Отже, $ord(2) = 10 = |Z_{11}^*|$. Це означає, що число α згенерувало всі елементи групи, отже воно є генератором. Група є циклічною.

Візьмемо інший приклад. Маємо групу $Z_7^* = \{1,2,3,4,5,6\}$. Хочемо перевірити чи $\alpha = 2$ є генератором в групі (див. таблицю 1.2).

Таблиця 1.2 – Перевірка генератора

Степінь	Обчислення	Результат
2^1	$2\ mod\ 11$	2
2^2	$4\ mod\ 11$	4
2^3	$8\ mod\ 11$	1

Тут є лише 3 різні результати замість 6. Отже, $\mathrm{ord}(2) = 3 < |Z_{11}^*|$. 2 не є генератором у Z_7^*. Група теж циклічна, але число 2 не є генератором групи.

Також існують нециклічні скінченні групи, тобто групи з обмеженою кількістю елементів, у яких жоден елемент не здатен утворити всі інші елементи групи. Наприклад, маємо групу $Z_1 \times Z_2$, що складаються з елементів $\{(0;0),(0;1),(1;0),(1;1)\}$ з покомпонентним додаванням по модулю 2. У цій групі жоден елемент не може згенерувати всі інші елементи, отже вона не є циклічною. Такі групи мають іншу структуру і не використовуються в алгоритмах на основі DLP, проте є важливими в загальній алгебраїчній теорії.

Також варто пам'ятати про мультиплікативну групу – група, де операцією є множення. Нехай, маємо Z_p^*, яка складається з усіх цілих чисел від 1 до $p-1$, p - просте число. Операція в цій групі – множення по модулю p. Група є скінченною і, як правило, циклічною. Саме в таких групах формується класична задача дискретного логарифму [4].

У криптографії, зокрема в алгоритмах з відкритим ключем, ми працюємо саме з скінченними циклічними групами, оскільки вони дозволяють побудувати функцію, яку легко обчислити, але складно обернути. Ця фундаментальна властивість односторонніх функцій робить можливим існування таких алгоритмів як Diffie-Hellman, ElGamal. Саме завдяки властивостям алгебраїчних структур є можливим побудувати обчислювально ефективні та стійкі до атак криптосистеми. Ретельне математичне розуміння різних груп є критично важливим для аналізу стійкості криптосистем, вибору параметрів виявлення потенційних вразливостей.

1.3 Проблема дискретного логарифму

Одним із ключових методів захисту даних є асиметрична криптографія. Основна ідея полягає у використанні пари ключів: відкритого – для шифрування повідомлення, закритого – для розшифрування. Найважливішою властивістю таких систем є односторонність математичних функцій, тобто можливість легко виконувати пряме перетворення (наприклад, знайти $\beta = g^x mod\ p$) але дуже складно обернене (знайти x знаючи β). Це і забезпечує нам надійність: навіть, якщо хтось перехопить наше зашифроване повідомлення та відкритий ключі, то

він не зможе отримати початкові дані без знання секретного ключа. Однією з таких односторонніх функцій є піднесення до степеня за модулем – обчислення виразу $g^x mod\, p$, де $\beta = g^x mod\, p$. В цьому контексті якраз і з'являється проблема дискретного логарифму. Вона названа аналогічно до звичайного логарифму в алгебрі, де ми розв'язуємо наступне рівняння (див. формулу 1.1):

$$a^x = b, \rightarrow x = \log_a b \tag{1.1}$$

Тоді як задача дискретного логарифмування у мультиплікативній циклічній групі лишків за модулем простого числа Z_p^* виглядає так: нехай p – просте число, α - первісний елемент в циклічній групі Z_p^*, а β – інший елемент з Z_p^*. Задача DLP полягає у знаходженні цілого числа $x, 1 \leq x \leq p-1,$ такого що (див. формулу 1.2):

$$a^x = \beta\ mod\ p \tag{1.2}$$

Таке ціле число x завжди існує, оскільки α є первісним елементом, і кожен елемент групи може бути виражений як степінь будь-якого первісного елемента. Це число x називається дискретним логарифмом β за основою α.

Щоб краще зрозуміти, як виглядає дана функція, розглянемо графік, який зображено на рисунку 1.1:

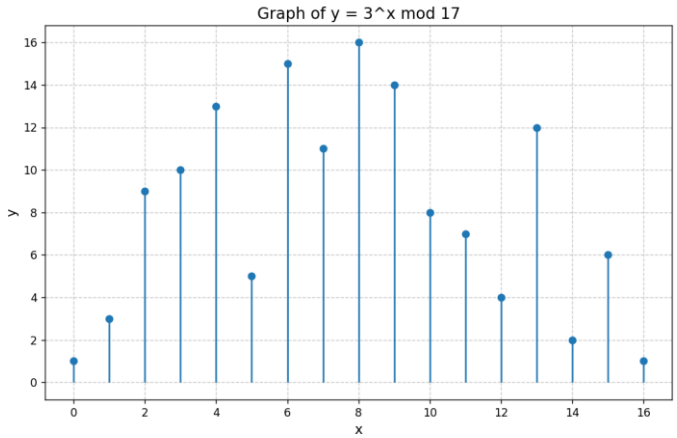

Рисунок 1.1 – Проблема дискретного логарифму (графічно)

Порівнюючи зі звичайною експоненціальною функцією, значення тут не ростуть монотонно, а коливаються в межах $[1, p-1]$. Саме така непередбачуваність значень, робить складним зворотне обчислення, це не можливо передбачити чи вгадати.

Використання цієї складності лежить в основі алгоритму обміну ключами Діффі-Хеллмана – одного з перших і найвідоміших застосувань дискретного логарифму в криптографії. У цьому протоколі дві сторони домовляються про спільне велике просте число p і генератор g. Кожна сторона генерує свій секретний ключ a, b, обчислює A $= g^a mod\ p$, B $= g^b mod\ p$, обмінюються цими значеннями. Потім обидві сторони можуть незалежно обчислити спільний секрет за формулою 1.3 [5]:

$$K = B^a mod\ p$$
$$K = A^b mod\ p \qquad (1.3)$$

Щоб краще зрозуміти, як відбувається обмін ключами, розглянемо рисунок 1.2, на якому ілюстровано алгоритм обміну ключами Діффі-Хеллмана. Тут показано, як учасники передають один одному відкриті значення та незалежно формують однаковий спільний ключ, який може бути використаний для подальшого шифрування. Рисунок ілюструє, що навіть при перехопленні відкритих значень третя сторона не зможе обчислити секретний ключ без розв'язання задачі дискретного логарифму.

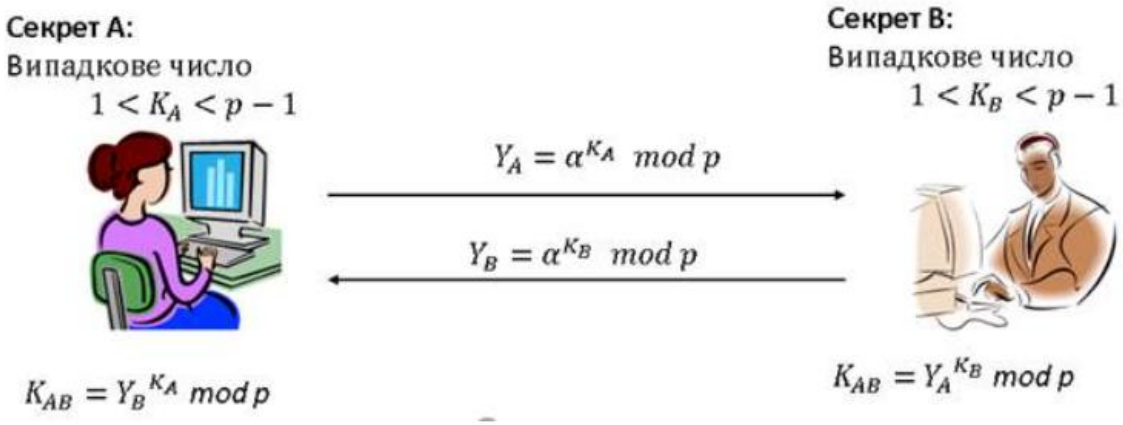

Рисунок 1.2 – Робота протоколу Діффі-Хеллмана [5]

Окрім класичної DLP у мультиплікативній групі Z_p^*, варто згадати задачу дискретного логарифма на еліптичних кривих (англ. Elliptic Curve Discrete Logarithm Problem, ECDLP). Дано еліптичну криву E, визначену над скінченним полем. ECDLP формулюється як знаходження цілого числа k, $1 \leq k \leq p-1$, якщо відомо дві точки P, Q на еліптичній кривій над скінченним полем, такі що (див. формулу 1.4):

$$Q = k * P \tag{1.4}$$

де число k – дискретний логарифм Q з основою P, P – базова точка на кривій, генератор. Тому:

$$k = \log_P Q \tag{1.5}$$

Знайти Q при відомих параметрах обчислювально легко, проте зворотна задача для знаходження k є значно складнішою, тобто є односторонньою функцією [6].

Розглянемо на рисунку 1.3 протокол Діффі-Хеллмана на еліптичних кривих. Цей протокол за своєю суттю аналогічний протоколу Діффі-Хеллмана, який базується на класичній задачі дискретного логарифму. Основна перевага такої реалізації – це вища крипостійкість при меншій довжині ключа.

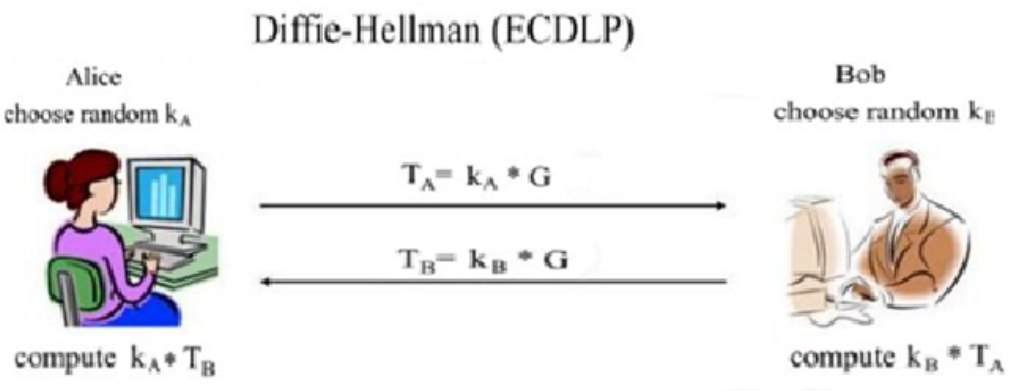

Рисунок 1.3 – Робота протоколу Діффі-Хеллмана (ECDLP) [7]

Задача дискретного логарифму на еліптичних кривих набагато важча для зламу, ніж звичайна. Тому для тієї ж безпеки в еліптичній криптографії можна використовувати значно коротші ключі, наприклад, 256 біт замість 3072. А ще для еліптичних кривих не існує швидких атак, які є для звичайної криптографії, що робить їх більш надійними та зручними.

РОЗДІЛ 2 АНАЛІЗ АТАК НА КРИПТОГРАФІЧНІ СИСТЕМИ, ЩО ПОБУДОВАНІ НА ЗАДАЧІ ДИСКРЕТНОГО ЛОГАРИФМУ

2.1 Загальна характеристика

Незважаючи на свою обчислювальну складність, задача дискретного логарифму не є абсолютно стійкою до атак. Розуміння типів атак на DLP є ключовим для правильного вибору криптографічних систем та оцінки їх безпеки. Для цього спочатку варто зрозуміти "слабкі місця" нашої задачі.

Проаналізувавши розділ 1, можна зрозуміти, що DLP не є універсально складною, її стійкість залежить від того, в якій групі обчислюється логарифм. Наприклад, якщо група має невеликий порядок, то можна просто перебрати всі можливі значення x.

Атаки на дискретний логарифм поділяють на загальні (англ. generic) та структурно-залежні (англ. non-generic) алгоритми. Загальні алгоритми працюють незалежно від структури групи, їх можна назвати універсальними.

До найвідоміших загальних атак належать:

- Метод повного перебору (англ. Brut-force) – метод перебору усіх можливих варіантів.
- Алгоритм великих малих кроків/метод Шенкса (англ. Baby-Step Giant-Step/Shanks method) – розбиває задачу на менші групи (розбиває x на два доданки), чим пришвидшує розв’язок задачі.
- Алгоритм Полларда "По" (англ. Pollard’s Rho algorithm) – ймовірнісний метод, який використовує випадкові зіткнення в послідовності елементів.

Ці методи показують ефективність і результат тоді, коли ми погано обрали групу або при недостатньо великих значеннях p.

Структурно-залежні алгоритми працюють на основі структури групи. Тобто, вони використовують розклад порядку групи на прості множники або властивості полів.

Серед відомих методів є:

• Алгоритм Поліг-Хеллмана (англ. Pohig-Hellman algorithm) – поділ задачі на менші для кожного простого дільника порядку групи, а потім об’єднує розв’язки через використання Китайської теореми про залишки.

• Алгоритм індексного числення (англ. Index Calculus) – будує систему рівнянь, де представляє елементи як добутки простих базисних чисел і розв’язує цю систему [1].

Усі ці методи мають різних принцип дії, переваги та недоліки. Для кращого розуміння розглянемо порівняльну таблицю основних алгоритмів, що використовують для розв’язання задачі дискретного логарифму (див. таблицю 2.1):

Таблиця 2.1 – Порівняльна характеристика

Алгоритм	Переваги	Недоліки	Часова складність	Пам’ять	Особливості
Метод повного перебору	Простота реалізації	Не практично, довго, неможливо для великих груп	O(p), x від 0 до $p-1$	Низька	Перебір усіх можливих значень x
Метод Шенкса	Зниження часової складності у порівнянні з наївним методом	Високі вимоги до пам’яті	$O(\sqrt{p})$	Висока	Порівняння значень із двох множин (малих і великих кроків)
Алгоритм Полларда "По"	Економія пам’яті порівняно з методом Шенкса	Залежність від випадкових траєкторій, складний аналіз	$O(\sqrt{p})$	Середня	Використо-вує принцип дня народження для пошуку колізій.
Індексна арифметика	Швидкий пошук у таблиці індексів	Великі вимоги до обчислюва-льних ресурсів	$O(\text{plog}(p))$	Дуже висока для побудови таблиці.	Попереднє обчислення індексів для фіксованих груп

Продовження таблиці 2.1

Алгоритм Pohlig-Hellman	Ефектив-ність у групах із малими простими множникам и	Залежність від найбільшого простого множника порядку групи	$O(\log(p) * q)$	Залежит ь від підзадач	Розбиває задачу на менші підзадачі за допомогою факторизаці ї порядку

Отже, атаки на дискретний логарифм можна звести до двох великих класів: універсальні, що завжди зменшують складність ціною різного обсягу пам’яті, та структурно-залежні, які "стріляють" миттєво, щойно порядок групи має зручний розклад або поле допускає субекспоненційний індекс-калькулюс. Перші упираються у квадратний корінь від розміру групи й виграють часово, коли доступна достатня оперативна пам’ять. Другі експлуатують факторизацію чи спеціальні властивості поля й різко прискорюють розв’язок, але лише за наявності відповідної структури. Таким чином, атака буде ефективною лише тоді, коли група "дозволяє" легко її зламати, або через багато варіантів, які можна швидко перебрати, або через зручну для аналізу структуру.

2.2 Загальні алгоритми

Метод повного перебору – це найпростіший метод або як його ще називають, метод грубої сили, основна ідея якого полягає в послідовному перебору усіх можливих значень x і перевірці рівності. Для кожного нового значення x обчислюється рівність (див. формулу 2.1), після чого результат порівнюється з . β

$$\alpha^x \bmod p \qquad (2.1)$$

Якщо значення не збігаються, до x додається одиниця і процес знову повторюється. Таке обчислення триватиме доти, поки не буде знайдено таке x, яке задовільняє рівняння і є розв’язком задачі дискретного логарифму [8].

Алгоритм Шенкса – це метод обміну між часом і пам’яттю, який зменшує час грубого пошуку за рахунок додаткового зберігання. Метод не залежить від структури групи, в якій обчислюється логарифм, і працює лише з порядком групи, тому його можна застосовувати до будь-яких DLP-систем. Метод малих і великих кроків зменшує простір перебору через розбиття x (див. формулу 2.2), де $m=\sqrt{p}$, а $i, j \in N$, i – маленький крок, j – великий крок:

$$x = i * m + j \tag{2.2}$$

Спершу, для кожного значення від $j = 0$ до $m - 1$ ми обчислюємо малі кроки за формулою 2.3 і всі значення зберігаються у хеш-таблиці або списку для швидкого пошуку:

$$L[i] = g^i \bmod p \tag{2.3}$$

Запис у таблицю дозволяє надалі швидко перевірити наявність збігів при виконанні великих кроків. Наступним кроком є обчислення оберненого елемента $y = g^{-m} \bmod p$, використовуючи розширений алгоритм Евкліда. Це обчислення нам необхідне для визначення великих кроків. Тому далі для кожного $i = 0$ до $m - 1$ обчислюємо великі кроки за формулою 2.4:

$$y * (g^{-m})^i \bmod p \tag{2.4}$$

Якщо результат присутній у таблиці L[i] , тобто якщо він збігається з одним із попередньо збережених малих кроків, то, вважаємо, що відповідь знайдено і останнім кроком буде(див. формулу 2.5) [9]:

$$x = i * m + j \tag{2.5}$$

Отже, метод Шенкса дозволяє значно зменшити час знаходження дискретного логарифму за умови наявності достатнього обсягу пам’яті для

збереження результатів попередніх обчислень. Завдяки своїй відносній простоті та ефективності метод часто використовується як базовий критерій для оцінки криптостійкості систем, побудованих на задачі дискретного логарифму.

Алгоритм Полларда "По", як і метод Шенкса, має таку ж очікувану складність виконання, але вимагає менше пам’яті. Метод є ймовірнісним, який працює за принципом створення послідовності елементів групи до моменту, коли не виникне зіткнення, тобто два однакових значення. Це ще називається принципом дня народження. Цей алгоритм створює послідовність, а коли в послідовності значення з’являється двічі, то з цього можна обчислити дискретний логарифм.

Починається алгоритм з обирання початкової стартової точки для побудови послідовності, зокрема (див. 2.6):

$$x_0 = 1, a_0 = 0, b_0 = 0 \tag{2.6}$$

Далі вся група умовно розбивається на три підмножини, наприклад, за залишком від ділення $x \bmod 3$. В залежності від того, в яку підмножину потрапляє поточне значення x_i, наступний елемент x_{i+1} обчислюється за одним з наступним правил:

- Якщо $x \bmod 3 = 0 \rightarrow x_{i+1} = x_i * y \bmod p$.
- Якщо $x \bmod 3 = 1 \rightarrow x_{i+1} = x_i^2 \bmod p$.
- Якщо $x \bmod 3 = 2 \rightarrow x_{i+1} = x_i * g \bmod p$.

Це дозволяє створити досить непередбачувану, але контрольовану послідовність чисел. І коли вперше з’явиться повторення значення в цій послідовності, тоді ми зможемо використати це повторення, щоб обчислити шуканий логарифм. Тобто, якщо ми обчислюємо нові значення крок за кроком, то рано чи пізно буде повторення. Ми робимо дві послідовності одночасно: x_i, x_{2i}. Кожна точка в послідовності має форму (див. формулу 2.7):

$$x_i = g^{a_i} * y^{b_i} \bmod p \tag{2.7}$$

Для ефективного виявлення зіткнення (тобто однакових значень x_i), застосовується так званий метод "черепахи і зайця", у якому одночасно будуються дві послідовності: одна зі звичайним кроком (черепаха) – x_i, а інша з подвійним (заєць) – x_{2i}. Зіткнення відбувається тоді, коли $x_i = x_{2i}$, що означає (див. формулу 2.8):

$$g^{a_i} y^{b_i} = g^{a_{2i}} y^{b_{2i}} \bmod p \tag{2.8}$$

Далі можна перетворити рівняння на формулу 2.9 через ділення обидвох частин на $g^{a_{2i}} y^{b_i}$:

$$g^{a_i - a_{2i}} = y^{b_{2i} - b_i} \bmod p \tag{2.9}$$

Підставляємо $y = g^x$ (див. формулу 2.10):

$$g^{a_i - a_{2i}} = g^{x(b_{2i} - b_i)} \pmod p \tag{2.10}$$

Переходимо до показників (формула 2.11):

$$a_i - a_{2i} \equiv x(b_{2i} - b_i) (\bmod p - 1) \tag{2.11}$$

Знаходимо x (див. формулу 2.12) при умові, що різниця коефіцієнтів b є взаємно простою з $p - 1$. Якщо цей найбільший спільний дільник відмінний від 1, оберненого елемента не існує, тому потрібно або повторити алгоритм, поки не отримаємо взаємно просту різницю, або розв’язати підсистему конгруенцій за допомогою розширеного алгоритму Евкліда/китайської теореми про лишки:

$$x = (a_i - a_{2i})(b_{2i} - b_i)^{-1} (\bmod p - 1) \tag{2.12}$$

Таким чином, алгоритм Полларда "По" – ефективно використовує властивості випадковості в циклічній групі для знаходження дискретного логарифму, витрачаючи при цьому мінімум пам’яті.

2.3 Спеціалізовані алгоритми

Index Calculus – це ефективний алгоритм для розв’язання задачі дискретного логарифму в великих простих полях, який використовує лінійну алгебру та факторизацію. Його суть – зібрати систему рівнянь, в яких логарифми малих простих чисел (factor base) виражаються через степені генератора. Після обчислення логарифмів елементів бази, шуканий логарифм цілі обчислюється через вже відомі значення.

На першому етапі обираємо факторну базу $B = \{p_1, p_2 \dots p_k\}$ – це набір невеликих простих чисел, якими буде зручно потім будувати рівняння. Далі виконується пошук експонент e, таких що значення $g^e \bmod p$ має повністю розкладатись на добуток простих із бази, тобто виконується наступна умова (див. формулу 2.13):

$$g^e \bmod p = p_1^{a_1} \cdot p_2^{a_2} \dots p_k^{a_k} \tag{2.13}$$

Беремо логарифм обох частин (див. формулу 2.14):

$$e = a_1 \log_g(p_1) + a_2 \log_g(p_2) + \dots + a_k \log_g(p_k) \bmod (p-1) \tag{2.14}$$

Це дає нам рівняння з k невідомими – логарифмами простих чисел факторної бази. Зібравши достатню кількість подібних рівнянь (не менше k), отримуємо систему рівнянь для $\log_g(p_i)$. Цю систему можна розв’язати класичними методами, наприклад, методом Гаусса в модульній арифметиці $(p-1)$. Далі обчислюємо логарифм цільового елемента. Для цього шукаємо таке e, що: $g^e y \bmod p$ повністю факторизується через факторну базу. Тобто виконується умова (див. формулу 2.15):

$$g^e y = p_1^{b_1} p_2^{b_2} \dots p_k^{b_k} \bmod p \tag{2.15}$$

Беручи логарифм двох частин, отримаємо (див. формулу 2.16):

$$\log_g(g^e y) = e + \log_g y = b_1 \log_g(p_1) + \dots + b_k \log_g(p_k) \bmod (p-1) \tag{2.16}$$

Звідси (див. формулу 2.17) [10]:

$$\log_g(y) = b_1 \log_g(p_1) + \dots + b_k \log_g(p_k) - e \bmod (p-1) \tag{2.17}$$

У результаті логарифм y обчислюється через уже відомі логарифми елементів факторної бази. Цей алгоритм значно пришвидшує розв'язання DLP у класичних полях. Індексне числення лежить в основі багатьох практичних атак на DLP у невдало сконфігурованих криптосистемах.

Наступним розглянемо алгоритм Pohlig-Hellman, який ґрунтується на факторизації порядку групи. На відміну від універсальних атак, цей алгоритм ефективно використовує внутрішню структуру групи. Його основна ідея полягає в тому, щоб розбити задачу на декілька підзадач, а потім працювати з кожною окремо. Далі результати об'єднуються за допомогою Китайської теореми про залишки (англ. Chinese theorem of remainder, CRT).

Китайська теореми про залишки – це спосіб розв'язувати задачі, коли потрібно знайти одне число, яке задовільняє кілька рівнянь, написаних у вигляді залишків (тобто в модульній арифметиці). Вона працює, якщо модулі в рівняннях є взаємно простими числами. Нехай задано k взаємно простих чисел $m_1, m_2, \dots, m_k$, тобто: НСД $(m_i, m_j) = 1$. Розглянемо систему конгруенцій (див. формулу 2.18):

$$\begin{aligned} x &= x_1 (\bmod m_1) \\ x &= x_k (\bmod m_k) \end{aligned} \tag{2.18}$$

Тоді:

- Існує єдиний розв’язок x у проміжку $0 \leq x < M$, де $M = m_1 * m_2 * m_k$.
- Цей розв’язок можна знайти за допомогою конструктивного методу.

Ключ до роботи CRT – властивість взаємної простоти модулів. Це означає, що будь-яке число , яке є рішенням одного рівняння, можна перевести до системи і результат буде єдиним у межах повного циклу.

Для знаходження x, яке задовольняє систему конгруенцій треба обчислити добуток модулів: $M = m1 \cdot m2 \ldots mk$. Далі для кожного рівняння знайдемо частковий модуль за формулою 2.19:

$$M_i = \frac{M}{m_i} \tag{2.19}$$

Знайдемо обернене число M_i^{-1}за модулем m_i . Обернене число M_i^{-1} задовольняє рівняння: $M_i * M_i^{-1} = 1 \bmod m_i$. Його можна знайти, використовуючи розширений алгоритм Евкліда. Після цього, обчислюємо часткові розв'язки. Для кожного рівняння обчислюється внесок у загальний розв’язок за формулою 2.20:

$$x_i = \alpha_i * M_i * M_i^{-1} \tag{2.20}$$

І тоді сума часткових розв’язків дає загальний розв’язок (див. формулу 2.21) [10]:

$$x = \sum_{i-1}^{k} x_i \bmod M \tag{2.21}$$

Результат x – це число, яке задовольняє всі конгруенції одночасно. В контексті криптографії це дає змогу "склеювати" розв’язки задачі дискретного логарифму.

2.4 Атаки на еліптичні криві

Хоч і проблема дискретного логарифму на еліптичних кривих вважається складнішою для розв’язання порівняно зі звичайним DLP у мультиплікативних групах, існують певні атаки, що дозволяють скоротити час обчислення логарифма в специфічних випадках. Еліптична криптографія значно виграє у співвідношенні "розмір ключа / рівень безпеки", оскільки наразі для неї не існує відомих субекспоненційних алгоритмів розв’язання, подібних до Index Calculus. Наприклад, 256-бітна еліптична крива забезпечує рівень безпеки, еквівалентний 3072-бітному модулю RSA. Тут ми теж можемо говорити про загальні і структурні атаки. До загальних атак, так само, як і до DLP відноситься Brute-force, метод Шенкса, метод Полларда "По" [11].

До спеціалізованих атак, які застосовуються лише в певних умовах, належать:

- Атака Вейля (Weil descent attack) – використовується для особливих класів еліптичних кривих над розширеними полями і намагається звести ECDLP до DLP у групах іншого типу.
- Атака MOV (Menezes-Okamoto-Vanstone) – можлива у випадку, якщо криву можна зіставити з мультиплікативною групою через паринг (англ. pairing), зводячи ECDLP до DLP у більшому полі.
- Атака з використанням спеціальної структури кривої – наприклад, для кривих з малим коефіцієнтом хордового обчислення або аномальних кривих (анормальних кривих атака Сатоха-Аракі).
- Атака MOV (англ. Menezes-Okamoto-Vanstone) – це специфічна атака на ECDLP, яка працює для окремих типів кривих. Основна ідея - перенесення ECDLP у мультиплікативну групу розширеного поля, де можна використати алгоритм типу Index Calculus чи Pohlig-Hellman, що значно ефективніший.

Умови для успішної атаки:

- Порядок точки P має бути великим простим числом n, ця точка має лежати на кривій над F_q: $P \in E(F_q), ord(P) = n$
- Embedding degree k – мінімальне число k, для якого: $n , (q^k - 1)$

Це гарантує, що група E[n], множина точок порядку n може бути вкладена у мультиплікативну групу , тобто paring існує.

- $k \leq 20$.

Першим кроком є знаходження найменшого натурального числа k, для якого виконується умова, що $q^k = 1 \bmod n$, де q – характеристика поля, n – порядок підгрупи, в якій задано точку P. Тобто k – порядок кратності q за модулем n, що означає, що k є найменшим цілим числом, для якого $q^k - 1$ ділиться на n. Наприклад, якщо $q = 211, n = 19;\ 211^1 \bmod 19 = 2;\ 211^3 \bmod 19 = 1 \rightarrow k = 3$. Після цього використовується білінеарне спарювання, наприклад, спарювання Вейля або Тейта, яке перетворює точки на еліптичній криві у елементи мультиплікативної групи розширеного поля (див. формулу 2.22)

$$e(P, Q) = e(P, P)^x = h \tag{2.22}$$

Таким методом ми звели ECDLP до DLP, для якої вже можна застосовувати відомі методи розв'язання дискретного логарифму, зокрема алгоритм повного перебору, метод Шенкса, алгоритм Полларда "По" чи індексне числення. Загалом редукція через спарювання є ефективною лише для певних типі кривих, зокрема тих, що мають малий параметр кратності k, оскільки в іншому випадку DLP теж буде складно знайти [12].

2.5 Рекомендації щодо підвищення стійкості криптографічних систем

Задачі дискретного логарифму та дискретного логарифму на еліптичних кривих є фундаментальними в сучасній криптографії та використовуються в багатьох широко поширених криптографічних протоколах, таких як Diffie-Hellman, ElGamal, DSA, а також у протоколах безпечного обміну ключами, в електронному підписі, в інфраструктурі блокчейну (наприклад, Bitcoin) тощо. Хоча математичні задачі DLP та ECDLP вважаються обчислювально складними при відповідному виборі параметрів, злом таких систем можливий при

використанні ненадійних або недостатньо перевірених параметрів, а також при наявності помилок у реалізації алгоритмів.

З метою забезпечення високого рівня стійкості таких систем необхідно враховувати як обчислювальну складність атак, так і специфічні практичні вектори, пов’язані з реалізацією, обміном даними та захистом апаратного середовища. Розглянемо комплекс практичних рекомендацій, дотримання яких значно підвищує криптостійкість систем, що базуються на DLP або ECDLP.

• Використання криптографічно сильних розмірів ключів. Розмір ключа безпосередньо впливає на складність атак перебору, атак типу "Baby-step giant-step", алгоритму Полларда "По" та інших. Згідно з поточними рекомендаціями NIST, для класичної дискретної криптографії рекомендований мінімальний розмір модуля p – 2048 біт. Для систем із підвищеними вимогами до конфіденційності (наприклад, у банківській сфері або держустановах) рекомендовано використовувати модулі довжиною 3072 або навіть 4096 біт [13]. У випадку еліптичної криптографії така сама криптостійкість досягається вже при довжині ключа в 256 біт, що робить еліптичні криві більш привабливими з точки зору швидкодії та розміру даних. Це пов’язано з відсутністю субекспоненційних алгоритмів для ECDLP – на відміну від DLP, де застосовуються, наприклад, методи індексного числення. Приклад: крива secp256r1 (NIST P-256) із 256-бітним ключем використовується у більшості державних криптосистем США та вважається безпечною для використання до появи квантових комп’ютерів [14]. У сфері відкритих стандартів також активно використовується Curve25519, яка забезпечує високий рівень безпеки, ефективність та захист від сторонніх каналів витоку [15].

• Уникання груп зі слабкою структурою. Алгоритми типу Pohlig-Hellman ефективно атакують групи, де порядок (тобто p−1 для DLP) розкладається на малі прості множники. Якщо $p - 1 = 2,3,5,7,11 \ldots$, то час атаки зменшується. Тому рекомендується використовувати "безпечні прості числа" типу $p = 2q + 1$. де p, q – прості числа [10].

• Використання перевірених еліптичних кривих. У контексті еліптичних кривих безпека залежить не тільки від довжини ключа, а й від властивостей самої

кривої. Зокрема, існують атаки (атака MOV, атака Вейля, атаки на аномальні криві), які знижують складність задачі ECDLP або зводять її до задачі DLP у менш стійкій групі. Тому критично важливо використовувати лише стандартизовані криві, які пройшли багаторічну перевірку експертами з криптоаналізу. До таких кривих належать: Curve25519 (використовується у WireGuard, Signal, WhatsApp), secp256r1 (затверджена NIST для офіційного використання), secp256k1 (використовується у протоколах блокчейну, зокрема у Bitcoin) [14]. Треба уникати "домашньої генерації" кривих або вибору випадкових коефіцієнтів без глибокого аналізу їхньої стійкості, оскільки навіть дрібна вразливість у параметрах може призвести до критичного зламу.

• Безпечна генерація випадкових чисел. У багатьох алгоритмах на основі DLP та ECDLP (наприклад, цифровий алгоритм підпису) використовується випадкове значення k для генерації підпису. Якщо це значення вгадується або повторюється, зловмисник легко відновлює приватний ключ. Такі атаки вже були у реальному житті. Приклад: у 2010-х роках масовий злам DSA-підписів на мобільних пристроях був спричинений повторним використанням однакового nonce k у підписах [16]. Тому потрібно: використовувати криптографічно безпечні генератори випадкових чисел (наприклад, `secrets`, `os.urandom`, `SystemRandom` у Python); не використовувати загальні функції на зразок `random.random()`; уникати ручного введення nonce або спрощених алгоритмів генерації.

• Захист від сайд-канальних атак. Сайд-канальні атаки не атакують сам алгоритм, а вивчають непрямі прояви його роботи – наприклад, час обчислень, споживання енергії, електромагнітні випромінювання або навіть звук роботи процесора. Особливо вразливими є смарт-карти, вбудовані пристрої, інтернет речей (англ. Internet of Things, IoT). Для захисту: усі обчислення повинні виконуватись у константному часі; бажано застосовувати маскування змінних, введення фіктивних операцій, додаткових шумів; у апаратних реалізаціях використовуються спеціальні мікросхеми з електромагнітним екрануванням та захищеним кешем [17].

• Перевірка параметрів під час обміну ключами. У протоколах на основі DLP та ECDLP важливо перевіряти, що отримані значення: не дорівнюють 1 або 0; дійсно належать допустимій групі або кривій; мають правильний порядок. Приклад: у Diffie-Hellman можлива атака, якщо зловмисник надішле $g^0 = 1$, і обчислений ключ буде відомим.

• Підготовка до постквантової криптографії. Алгоритм Шора, що виконується на квантовому комп'ютері, дозволяє ефективно розв'язувати задачі факторизації, DLP і ECDLP, зводячи їхню складність до поліноміальної [18]. Це означає, що практично всі класичні криптосистеми будуть зламані у квантовому майбутньому. Хоча повноцінних квантових комп'ютерів наразі не існує, вже зараз активно розробляються гібридні криптографічні схеми, які поєднують класичні та постквантові алгоритми. Приклад: поєднання ECDH (англ. Elliptic Curve Diffie-Hellman) (Curve25519) з алгоритмом Kyber (переможець фіналу конкурсу NIST Post-Quantum Cryptography) дозволяє захиститися як від класичних, так і від квантових атак [19].

Забезпечення надійної криптографічної безпеки – це не тільки вибір складного алгоритму, а й увага до деталей: правильний вибір параметрів, безпечна реалізація, захист від побічних каналів і дотримання сучасних стандартів. DLP- і ECDLP-системи можуть залишатись надійними протягом десятиліть за умови грамотної інтеграції в програмно-апаратне середовище та вчасного реагування на нові загрози.

РОЗДІЛ 3 ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ АТАКИ ПОЛІГ-ХЕЛЛМАНА

3.1 Математична реалізація

На рисунку 3.1, який наведено нижче показано головну ідею алгоритму Pohlig-Hellman.

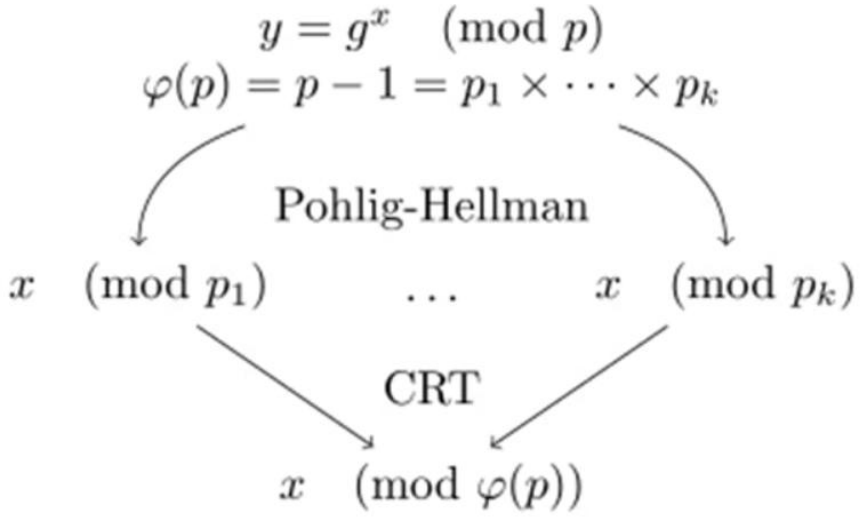

Рисунок 3.1 – Алгоритм Pohlig-Hellman [20]

Надалі здійснимо більш глибокий аналіз цього алгоритму. Розглянемо його ключові етапи та принципи роботи детальніше.

Вхідні дані: α: генератор (основа) модульного степеня; β: значення $\beta = \alpha^x \bmod p$; p: просте число; x: дискретний логарифм. Потрібно знайти x, таке що задовільняє рівняння $\alpha^x = \beta \bmod p$.

Спершу розкладаємо p−1 на прості множники за формулою 3.1, де q_i- прості числа, e_i – відповідні степені (див. 3.1).

$$p - 1 = q_1^{e1} * q_2^{e2} * \ldots * q_k^{ek} \quad (3.1)$$

Далі обчислюємо окремі x_i для кожного $q_i^{e_i}$. Розв'язуємо задачу дискретного логарифма по модулю , $q_i^{e_i}$, тобто знайти x_i, який задовольняє наступну нерівність (див. формулу 3.2):

$$\beta^{\frac{p-1}{q_i^{e_i}}} = \alpha^{\frac{(p-1)x_0}{q_i^{e_i}}} \bmod p \tag{3.2}$$

Ми виконуємо це поступово, розкладаючи x_i за базою q_i (див. формулу 3.3)

$$x_i = x_{i,0} + x_{i,1} * q_i + x_{i,2} * q_i^2 + \ldots + x_{i,e-1} * q_i^{e-1} \tag{3.3}$$

Знаходимо кожний $x_{i,j}$ використовуючи ітеративний підхід (у нашому випадку метод перебору) або за допомогою інших алгоритмів для малих підгруп. Отримані значення x_i для всіх і формують систему конгруенцій (див 3.4).

$$\begin{aligned} x &= x_1 (\bmod q_1^{e_1}) \\ x &= x_2 (\bmod q_2^{e_2}) \\ x &= x_k (\bmod q_k^{e_k}) \end{aligned} \tag{3.4}$$

Цю систему конгруенцій розв’язуємо за допомогою Китайської теореми про залишки, що дозволяє зібрати всі окремі розв’язки x_i у єдиний розв’язок $x \bmod (p-1)$, як показано в 3.5.

$$\begin{cases} x = r_1 (\bmod \alpha_1) \\ x = r_2 (\bmod \alpha_2) \\ x = r_n (\bmod \alpha_n) \end{cases} \tag{3.5}$$

Отримуємо очікуваний результат (див формулу 3.6) [21]

$$\beta = \alpha^x \bmod p \tag{3.6}$$

Розглянемо приклад:

Вхідні дані: $\alpha = 7$ – генератор (основа) модульного степеня, $\beta = 12$ – значення $\beta = \alpha^{x} \bmod p$, $p = 41$ – просте число. Потрібно знайти x, таке що задовільняє рівняння $\alpha^{x} = \beta \bmod p$

Першим кроком розкладаємо $p-1$ на прості множники. За допомогою послідовного ділення отримаємо: $40 = 2 * 2 * 2 * 5 = 2^3 * 5$. Отже, порядок групи має два прості множники $q = (2, 5)$, для кожного з яких потрібно знайти відповідний розв'язок $x \bmod q_i^{e_i}$, після чого об'єднати результати через Китайську теорему про залишки.

Розв'язуємо задачу дискретного логарифма, тобто треба знайти x_i, який задовольняє наступне (див формулу 3.7):

$$\beta^{\frac{p-1}{q_i^{e_i}}} = \alpha^{\frac{(p-1)x_0}{q_i^{e_i}}} \bmod p \tag{3.7}$$

Тепер обчислюємо окремі x_i для кожного $q_i^{e_i}$.

Коли $q = 2$, $x = x_0 + 2^1 x_1 + 2^2 x_2$, ми використовуємо наступну формулу (див. 3.8):

$$x_0 : \beta^{\frac{p-1}{q_0}} = (\alpha^{\frac{p-1}{q}})^{x_0} \tag{3.8}$$

Підставлячи, ми отримуємо: $12^{\frac{40}{2}} = (7^{\frac{40}{2}})^{x_0} \bmod 41 \Longrightarrow 12^{20} = (7^{20})^{x_0} \bmod 41 \Rightarrow -1 \bmod 41 = (-1)^{x_0} \bmod 41$. Виконаємо метод перебору: $-1 \bmod 41 \neq (-1)^0 \bmod 41$. $-1 \bmod 41 = (-1)^1 \bmod 41$. Отже, $x_0 = 1$.

Аналогічно знаходимо x_1 за наступною формулою (див. формулу 3.9)

$$x_1 : \beta_1^{\frac{p-1}{q_1}} = (\alpha^{\frac{p-1}{q}})^{x_1} \tag{3.9}$$

У нас, $q_1 = 2^2$. Далі формуємо допоміжний елемент $\beta_1 = \beta * \alpha^{-(x_1)} \Longrightarrow 12 * 7^{(-1)}$

Згідно з алгоритмом Евкліда розв'язуємо : $41 = 5 * 7 + 6, 7 = 1 * 6 + 1, 6 = 6 * 1 + 0$. НСД (7, 41)=1 взаємно прості, обернений елемент існує. Згідно з розширеним алгоритмом Евкліда: $7x + 41y = 1$, $7^{-1} = 6 \bmod 41$, $x = 6$. Відповідно β_1 шукаємо наступним чином.: $\beta_1 = 12 * 6 \bmod 41 = 72 \bmod 41 = 31$.Тоді x_1: x_1: $31^{\frac{40}{4}} = (7^{\frac{40}{2}})^{x_1} \bmod 41 \Longrightarrow 31^{10} = (7^{20})^{x_1} \bmod 41 \Rightarrow 1 \bmod 41 = (7^{20})^{x_1} \bmod 41$.Так, як ліва частина дорівнює 1, то права теж повинна бути їй рівна. Це можливо тільки якщо $x_1 = 0$. Отже, $x_1 = 0$.

Шукаємо x_2 за наступною формулою (див 3.10):

$$x_2: \beta_2^{\frac{p-1}{q_2}} = (\alpha^{\frac{p-1}{q}})^{x_2} \tag{3.10}$$

У нас $q_2 = 2^3$. $\beta_2 = \beta_1 * \alpha^{-(x_2)} \Longrightarrow 31 * 7^{-0} = 31 \bmod 41$. Знаходимо x_2 : $x_2: 31^{\frac{40}{8}} = (7^{20})^{x_2} \bmod 41 \Rightarrow -1 \bmod 41 = -1^{x_2} \bmod 41$, $x_2 = 1$

Тепер шукаємо x за наступною формулою (див 3.11):

$$x = x_0 + 2^1 x_1 + 2^2 x_2 \tag{3.11}$$

Підставляємо: $x = 1 + 2^1 * 0 + 2^2 * 1 = 5$ $x = 5 \bmod 2^3 = 5 \bmod 8$

Коли $q = 5$, $x = 5^0 x_0$, то (див 3.12):

$$x_0: \beta^{\frac{p-1}{q_0}} = (\alpha^{\frac{p-1}{q}})^{x_0} \tag{3.12}$$

Підставляємо: $12^{\frac{40}{5}} = (7^{\frac{40}{5}})^{x_0}$, $12^8 \bmod 41 = 18$, $7^8 \bmod 41 = 37$. Використаємо метод перебору : $18 \neq 37^0 \bmod 41$ $18 \neq 37^1 \bmod 4118 \neq 37^2 \bmod 41$ $18 = 37^3 \bmod 4$ Отже, $x_0 = 3$. Звідси знаходимо x: $x = 5^0 x_0$ $x = 5^0 * 3 = 3$ $x = 3 \bmod 5$

Тому, $x = 5 \bmod 8$, $x = 3 \bmod 5$.

Розв’язуємо систему конгруенцій за допомогою Китайської теореми про залишки. Примітка: модулі 8 і 5 взаємно прості (НСД(8,5)=1), тому система має розв’язок. Для цього ми знаходимо m_1 і m_2 (див.3.13):

$$m_1 = \frac{p-1}{8} = \frac{40}{8} = 5 \qquad (3.13)$$
$$m_2 = \frac{p-1}{5} = \frac{40}{5} = 8$$

Знаходимо обернені елементи (див. 3.14):

$$m_1^{-1} = 5 * x = 1 \bmod 8 \Rightarrow x = 5 \qquad (3.14)$$
$$m_2^{-1} = 8 * x = 1 \bmod 8 \Rightarrow x = 2$$

Обчислюємо розв’язок за формулою 3.15:

$$x = \alpha_1 * m_1 * m_1^{-1} + \alpha_2 * m_2 * m_2^{-1} \bmod p \qquad (3.15)$$

Підставляємо: $x = 5 * 5 * 5 + 3 * 8 * 2 \bmod 40$, $x = 173 \bmod 40$ $173 \bmod 40 = 13, x = 13 \bmod 40$. Відповідь: 13.

3.2 Програмна реалізація

Створюючи код для Pohlig-Hellman, ми користувались тим самим алгоритмом, який описано у розділі 3.1. Ми використали мову програмування Python для реалізації цієї задачі, адже вона забезпечує зручний синтаксис, високу читабельність коду та має багато можливостей для роботи з математичними операціями, різними числовими алгоритмами. Python є популярним вибором у криптографічних дослідження через велику кількість вбудованих функцій та наявності спеціальних бібліотек

Ми використовували стандартну бібліотеку `math`, яка дозволяє виконувати математичні обчислення, а саме функції `sqrt` (для обчислення квадратного кореня) та `ceil` (для округлення вгору).

Для початку ми повинні розкласти число на прості множники q, які необхідні для формування конгруенцій (див. лістинг 3.1).

Лістинг 3.1 – Функція для розкладання на прості множники

```
def PrimeFactorization(p):
    d, primeFactors = 2, []
    while d * d <= p:
        while (p % d) == 0:
            primeFactors.append(d)
            p //= d
        d += 1
    if p > 1:
        primeFactors.append(p)
    return primeFactors
```

Далі за допомогою алгоритму Евкліда, знаходимо g (див. лістинг 3.2). Це застосовується для знаходження оберненого елемента та розв'язання конгруенцій.

Лістинг 3.2 – Розширений алгоритм Евкліда

```
def ExtendedGCD(a, b):
    a2, a1 = 1, 0
    b2, b1 = 0, 1
    while b:
        q, r = divmod(a, b)
        a1, a2 = a2 - q * a1, a1
        b1, b2 = b2 - q * b1, b1
        a, b = b, r
    return a, a2, b2
```

На основі алгоритму Евкліда, знаходимо обернений елемент, який потрібний для модульних обчислень (див. лістинг 3.3).

Лістинг 3.3 – Знаходження оберненого елементу $b^{-1} \bmod n$

```
def ModularInverse(b, n):
    g, x, _ = ExtendedGCD(b, n)
    if g == 1:
        return x % n
```

Китайська теорема залишків комбінує розв'язки окремих модулів у єдиний результат (див. лістинг 3.4).

Лістинг 3.4 – Китайська теорема залишків

```
def ChineseRemainder(pairs):
    N, X = pairs[0][1], 0
    for ni in pairs[1:]:
        N *= ni[1]
    for (ai, ni) in pairs:
        mi = (N // ni)
        X += mi * ai * ExtendedGCD(mi, ni)[1]
    return X % N
```

За допомогою кроків Шенкса знаходимо $\beta = \alpha^x$ (див. лістинг 3.5):

Лістинг 3.5 – Кроки Шенкса

```
def ShanksAlgorithm(alpha, beta, n):
    """Return x such that beta ≡ alpha^(x) (mod n)"""
    m = int(ceil(sqrt(n - 1)))
    a = pow(alpha, m, n)
    b = ExtendedGCD(alpha, n)[1]
    L1 = [(j, pow(a, j, n)) for j in range(0, m)]
    L2 = [(i, beta * (b ** i) % n) for i in range(0, m)]
    L1.sort(key=lambda tup: tup[1])
    L2.sort(key=lambda tup: tup[1])
    i, j, Found = 0, 0, False
```

```
    while (not Found) and (i < m) and (j < m):
        if L1[j][1] == L2[i][1]:
            return m * L1[j][0] + L2[i][0] % n
        elif abs(L1[j][1]) > abs(L2[i][1]):
            i += 1
        else:
            j += 1
```

Формуємо окремі конгруенції (x, q^e) для кожного простого множника q (див. лістинг 3.6).

Лістинг 3.6 – Окремі конгруенції

```
def CongruencePair(g, h, p, q, e, e1, e2):
    alphaInverse = ModularInverse(e1, p)
    x = 0 # x = x_{0} + x_{1} * q + x_{2} * q^{2} + ... + x_{e
- 1} * q^{e - 1}
    for i in range(1, e + 1):
        a = pow(e1, q ** (e - 1), p)
        b = pow(e2 * (alphaInverse ** x), q ** (e - i), p)
        x += ShanksAlgorithm(a, b, p) * (q ** (i - 1))
    return (x, q ** e)
```

Для кращого розуміння принципу роботи шифрування на основі дискретного логарифму, спершу скористаємось програмою для обчислення степеня за модулем, щоб отримати значення $h = g^x \bmod p$ (див. лістинг 3.7). На рисунку 3.2 показано використання цієї програми. Це імітує процес шифрування, де відоме значення g, секретний ключ x і просте число p. Отримане значення h є результатом шифрування. Далі, для перевірки правильності та демонстрації зворотного процесу, застосовуємо алгоритм Поліга-Хеллмана з метою відновлення значення x, тобто здійснюємо обернену операцію дешифрування.

Лістинг 3.7 – Обчисленя степеня за модулем

```
g = int(input("Enter the value of g: "))
```

```
x = int(input("Enter the value of x: "))
p = int(input("Enter the value of p (modulus): "))
h = pow(g, x, p)
print(f"Result: h = {h}")
```

```
PS C:\Users\Uliana\Downloads> python codeee.py
Enter the value of g: 7
Enter the value of x: 13
Enter the value of p (modulus): 41
Result: h = 12
PS C:\Users\Uliana\Downloads>
```

Рисунок 3.2 – Результат обчислення степеня за модулем

На рисунку 3.3 показано успішну реалізацію Поліг-Хеллмана. Цілісний код подано в додатку А. Як бачимо, результати програмної і математичної реалізацій Pohlig-Hellman точно збігаються, що підтверджує коректність кроків. Також значення x зійшлось з введеним значенням x (13), яке було показано на рисунку 3.2.

```
h = 12
g = 7
p = 41
Solving 12 = 7^x (mod 41)

For q, the factors of (p-1) are: [2, 2, 2, 5]
For factors, [q,e] prime, counts: [[2, 3], [5, 1]]


 q  | e  | g^((p-1)/q^e) | h^((p-1)/q^e) | (g^((p-1)/q^e))^x = h^((p-1)/q^e)
-----------------------------------------------
 2  | 3  | 38            | 3             | x =  5 (mod  8)
-----------------------------------------------
 5  | 1  | 37            | 18            | x =  3 (mod  5)
-----------------------------------------------
Solution x = 13
Checking h = 12
```

Рисунок 3.3 – Успішна реалізація

Розглянемо варіанти, коли Pohlig-Hellman не спрацює (див. рисунок 3.4). Як бачимо, p – не є простим числом, тому отримуємо помилку.

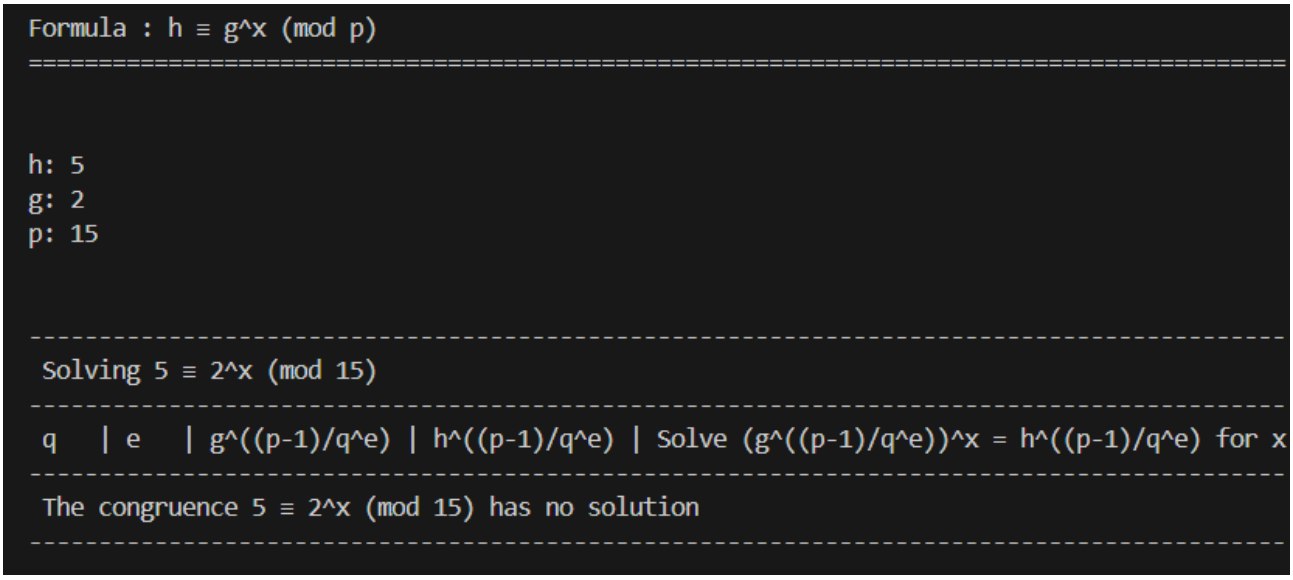

Рисунок 3.4 – Перевірка коду

Також ми не зможемо обчислити Pohlig-Hellman, коли g не є примітивний коренем по модулю p (див. рисунок 3.5):

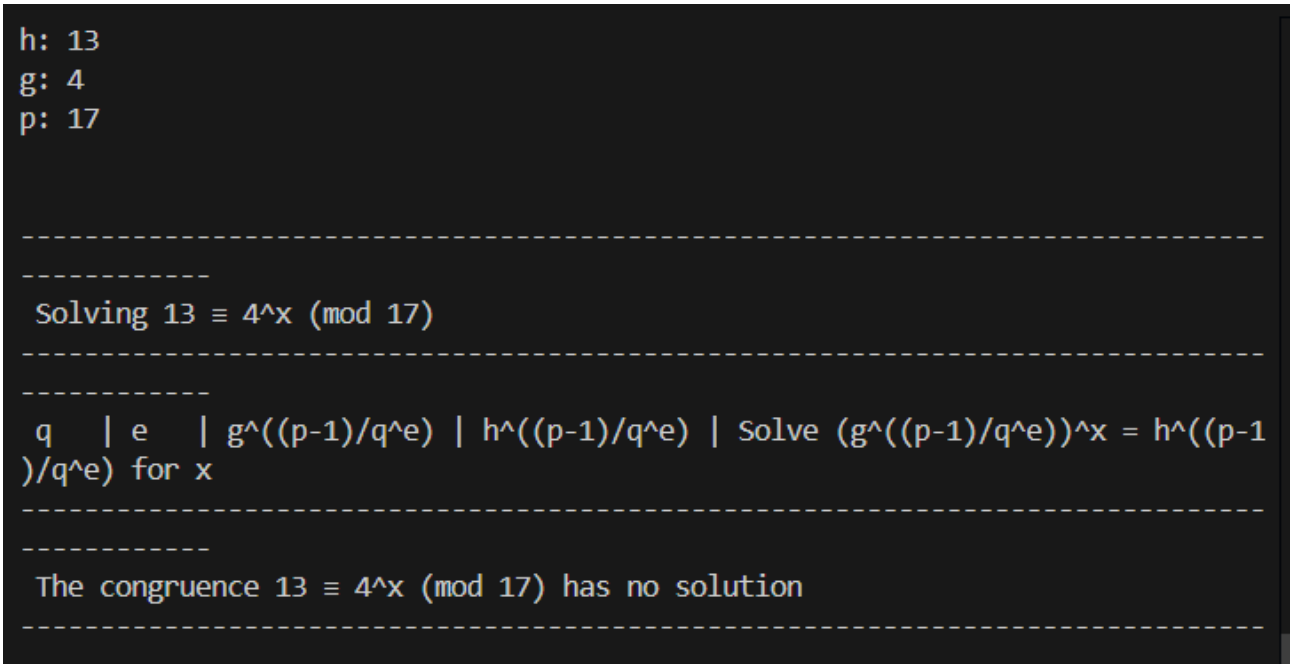

Рисунок 3.5 – Перевірка коду

При правильному виборі генератора, ми отримуємо результат (див. рисунок 3.6). Результат обчислення степеня за модулем показано на рисунку 3.7. Також значення x зійшлось з введеним його початковим значенням x=4.

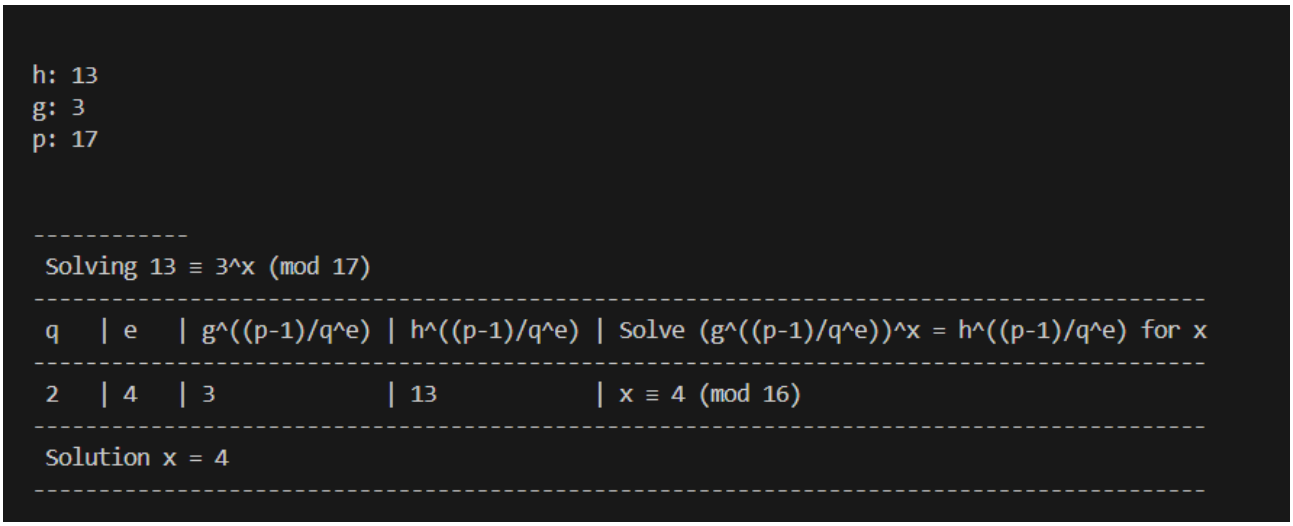

Рисунок 3.6 – Перевірка коду

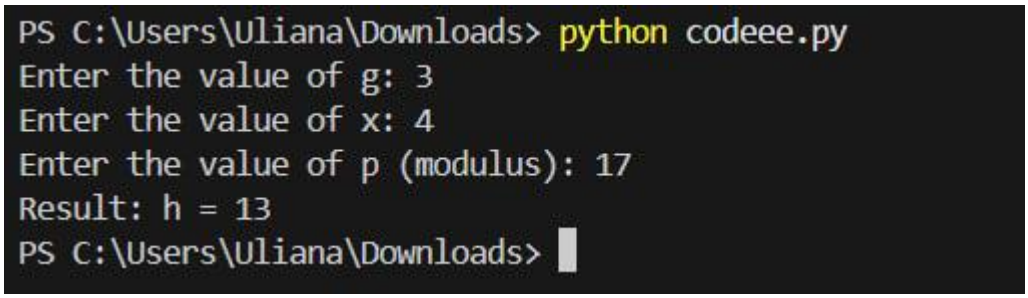

Рисунок 3.7 – Результат обчислення степеня за модулем

На рисунку 3.8 представлено приклад роботи алгоритму Поліг-Хеллмана з трьох- та чотиризначними числами, що демонструє коректність реалізації навіть при збільшенні розмірності вхідних даних. Результат обчислення степеня за модулем показано на рисунку 3.9. Також значення x зійшлось з введеним його початковим значенням x=1066.

```
h: 729
g: 2
p: 1223


-------------------------------------------------------------------------------------------------
 Solving 729 ≡ 2^x (mod 1223)
-------------------------------------------------------------------------------------------------
 q   | e   | g^((p-1)/q^e) | h^((p-1)/q^e) | Solve (g^((p-1)/q^e))^x = h^((p-1)/q^e) for x
-------------------------------------------------------------------------------------------------
 2   | 1   | 1             | 1             | x ≡ 0 (mod 2)
-------------------------------------------------------------------------------------------------
 13  | 1   | 706           | 1             | x ≡ 0 (mod 13)
-------------------------------------------------------------------------------------------------
 47  | 1   | 408           | 671           | x ≡ 32 (mod 47)
-------------------------------------------------------------------------------------------------
 Solution x = 1066
-------------------------------------------------------------------------------------------------
```

Рисунок 3.8 – Робота з більшими числами

```
PS C:\Users\Uliana\Downloads> python codeee.py
Enter the value of g: 2
Enter the value of x: 1066
Enter the value of p (modulus): 1223
Result: h = 729
PS C:\Users\Uliana\Downloads>
```

Рисунок 3.9 – Результат обчислення степеня за модулем

Програмна реалізація методу Поліг-Хелмана показала високу точність і коректність роботи, отримані результати збігаються з математичними обчисленнями.

3.3 Аналіз отриманих результатів

Після математичного опису та програмної реалізації алгоритму Pohlig-Hellman варто оцінити, наскільки коректно він працює за різних вхідних параметрів. Розглянемо таблицю 3.1:

Таблиця 3.1 - Результати тестування

Дані для експериментів						Результат
Приклад	Початкове значення x	p	p-1	g	Властивості g	
1	13	41	2*2*2*5	7	g є примітивним	Результат знайдено, x=13
2	2	15	7*2	2	g не є примітивним, група не циклічна	Результат не знайдено
3	8	17	2*2*2*2	4	g не є примітивним, $4^8 \neq 1 \bmod 17$	Результат не знайдено
4	4	17	2*2*2*2	3	g є примітивним	Результат знайдено, x=4
5	1066	1223	2*13*47	2	g є примітивним	Результат знайдено, x=1066

Аналізуючи приклад 1, коли модуль дійсно простий, а генератор є первісним елементом, реалізація алгоритму Pohlig-Hellman точно відтворює результат, який був у нас в математичній реалізації: часткові логарифми, знайдені у підгрупах порядку 8 і 5, успішно поєднуються Китайською теоремою про залишки й дають кінцеве значення x=13. Приклад 2 показує, що сама відсутність простоти модуля робить задачу некоректною: група Z_{15}^* не є циклічною, тому єдиного x не існує, і програма завершує роботу без розв'язку. У третьому експерименті модуль простий, але вибрано неправильний генератор, який не породжує всю групку порядку 16, тому ми не можемо це розв'язати. Після заміни на справжній примітивний корінь, алгоритм одразу знаходить x. Приклад 5 демонструє масштабованість, навіть при більшому модулі програма працює правильно і повертає x.

Усі ці спостереження підводять до задачі дискретного логарифму. Надійність DLP-схем визначається не абстрактною "складністю піднесення до степеня", а саме тим, у якій групі виконується операція. Якщо порядок групи легко факторизується або містить невеликі прості множники, атака Pohlig–

Hellman дробить задачу на підзадачі й різко знижує складність розв’язку; якщо ж сам модуль не простий або генератор не примітивний, то рівняння може взагалі не мати розв’язку. Лише вибір групи з великим первісним простим порядком, а також ретельна перевірка генератора повертають задачу до площини універсальних атак Шенкса чи Полларда "По" і залишається практично недосяжною за достатньо великих параметрів. Таким чином, експериментальна перевірка не лише підтвердила коректність програмної реалізації Pohlig-Hellman, а й наочно продемонструвала фундаментальний принцип сучасної криптографії: безпека схем, побудованих на дискретному логарифмі, критично залежить від грамотного вибору групових параметрів, які закривають усі відомі структурні вектори атак.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Працездатність людини – оператора

Людина-оператор – це користувач, який здійснює трудову (професійну) діяльність у системі "людина-техніка-середовище", активно взаємодіючи з технічними засобами керування чи моніторингу - наприклад, з комп’ютерною системою або пультом управління. Оператор сприймає інформацію, аналізує її та приймає рішення, керуючись результатами взаємодії з технічним обладнанням. Людина-оператор характеризується швидкодією, надійністю, точністю [22].

Працездатність – це можливість виконувати роботу якісно й у відведений час, залежно як від зовнішніх (умови праці, кількість інформації, дизайн робочого місця, взаємодія з колегами), так і від внутрішніх чинників (професіоналізму, фізичної підготовки, емоційної стійкості) .

Функціональна працездатність оператора змінюється протягом робочого періоду і складається з трьох фаз:

- Адаптації – період входу в темп.
- Стійкого стану – оптимальний рівень уваги, продуктивності.
- Субкомпенсації (втоми) – коли ресурси вичерпуються та зростає кількість помилок і знижується швидкість реакції [23].

Фаза адаптації (входження в робочий ритм) – це початкова стадія, яка характеризується поступовою адаптацією оператора до умов праці. У цей період активуються когнітивні та психофізіологічні механізми, відбувається налагодження взаємодії із системою. Швидкість реакцій може бути дещо зниженою, спостерігається пошук оптимальної стратегії виконання завдань, може бути підвищена тривожність. Тривалість фази адаптації залежить від складності завдання, досвіду оператора, мотивації та умов праці.

Фаза стійкого стану (оптимальна працездатність) – це основна, найпродуктивніша фаза, яка характеризується стабільним рівнем уваги, високою точністю дій, мінімальною кількістю помилок. Оператор працює в

оптимальному ритмі, добре координує дії, використовує автоматизовані навички. У цьому стані забезпечується максимальна ефективність та надійність управлінських рішень. За сприятливих умов ця фаза може тривати тривалий час. Для підтримки її максимальної тривалості необхідно:

- Забезпечити раціональний режим праці й відпочинку (регламентовані перерви, мікропаузи).
- Використовувати ергономічні засоби праці
- Підтримувати оптимальні умови мікроклімату (температура 20–24°C, вологість 40–60%).
- Забезпечити інформаційно-ергономічний інтерфейс для швидкого сприйняття сигналів.

Серед цих факторів варто згадати емоційну атмосферу в колективі. Дружнє середовище, підтримка з боку колег, справедливе керівництво та відсутність міжособистісних конфліктів сприяють зниженню емоційного напруження та підвищенню мотивації. Навпаки, психологічний дискомфорт, напруга у спілкуванні, відсутність зворотного зв'язку та несправедливість можуть призводити до тривожності, зниження концентрації уваги й підвищення кількості помилок. Для збереження оптимальної працездатності важливо не лише дотримуватись гігієнічних та ергономічних вимог, а й підтримувати позитивну соціально-психологічну атмосферу в команді, що забезпечує емоційну стабільність і підвищує загальну ефективність трудової діяльності.

Фаза субкомпенсації або втоми – це зниження працездатності Унаслідок виснаження ресурсів організму настає стадія зниження концентрації уваги, реакцій та точності дій. Можуть виникати помилки, затримки у прийнятті рішень, зростає потреба у фізіологічному та психологічному відновленні. Втома може проявлятися як фізичне виснаження, так і ментальне перенавантаження. Якщо оператор продовжує працювати в такому стані, це може призвести до зростання ризику аварійних ситуацій або зниження якості продуктивності.

Важливо знати, що зміна фаз працездатності є природним процесом. Проте завдання організації полягає у мінімізації часу адаптації, продовженні фази стійкої ефективності та запобіганні переходу до глибокої втоми. Працездатність

людини-оператора є ключовим чинником ефективності у взаємодії з технічними системами. Розуміння фаз її змін - адаптації, стійкого стану та субкомпенсації - дозволяє раціонально організувати умови праці та мінімізувати помилки. Забезпечення оптимального робочого середовища сприяє збереженню високої надійності, точності й безпеки операторської діяльності.

4.2 Системи протипожежної безпеки в Центрах обробки даних

Центр обробки даних (ЦОД) – це спеціалізовані інженерно-технічні об’єкти, які призначені для розміщення серверного, мережевого, телекомунікаційного та іншого високотехнологічного обладнання. Це є основа для функціонування цифрової інфраструктури різних державних органів, банків, хмарних середовищ тощо. Будь-яке порушення у роботі ЦОД, до прикладу пожежа, може призвести до втрати критичних даних, зупинки бізнес-процесів або навіть до інцидентів національного рівня. Такі події будуть супроводжуватись величезними фінансовими збитками. Саме тому системи протипожежної безпеки в ЦОД мають не тільки технічне, але і стратегічне значення.

Одними з основних причин пожеж у ЦОД є: коротке замикання в електроживленні серверів або шаф, перегрів обладнання, займання кабельних трас при перевантаженнях, тривале тління мікросхем або проводки тощо. Ці події можуть відбуватись в різних формах і вимагати різних підходів до реагування.

У приміщеннях з високою щільністю ІТ-обладнання рекомендовано застосовувати аспіраційні системи раннього виявлення диму, наприклад VESDA. Вони постійно відбирають проби повітря по всій серверній кімнаті через мережу труб і аналізують його за допомогою лазерних сенсорів на наявність найменших частинок диму. Це дозволяє виявити тління ще до появи візуального диму або полум’я. Також допускається комбінування з точковими димовими детекторами класу А для підвищення надійності. У випадку виявлення загоряння система сигналізації має бути підключена до системи керування інженерною інфраструктурою будівлі. Тобто при фіксації загрози автоматично передається

сигнал на диспетчерський пульт, активується сповіщення персоналу, відбувається блокування доступу до зони займання, а також запускається процес гасіння.

Використання води в серверних об'єктах не рекомендується, оскільки є ризик пошкодження електроніки. Тому основними засобами пожежогасіння в ЦОД є:

•Газові системи гасіння (включно з FM-200, Novec 1230, Inergen) – безпечні для ІТ-обладнання, які ефективно працюють за наступними принципами: ізоляції та інгібування. Ізоляції - заміщення кисню інертним газом (азот, аргон, інерген). Інгібування – гальмування хімічних реакцій у полум'ї галогенізованими речовинами. Вони не проводять електрику, не залишають осадів і є безпечними для обладнання.

• Аерозольні системи – застосовуються для невеликих замкнених приміщень, де треба швидка локалізація пожежі. Аерозоль створює дрібнодисперсну хмару, яка блокує доступ кисню до полум'я.

• Гіпоксичне середовище – підтримка низького рівня кисню за допомогою подачі азоту. Так атмосфера унеможливлює виникнення і поширення вогню.

• Вогнегасники CO_2 – розміщуються локально для ручного використання до або після активації автоматичних систем [24].

Крім самих систем виявлення та гасіння, важливе значення мають інженерні рішення, які запобігають поширенню вогню. Рекомендується створення вогнестійких бар'єрів, розділення приміщень на протипожежні відсіки, ізоляцію кабельних каналів, використання вогнестійких дверей з автоматичним закриттям, герметизацію проходів труб і кабелів. Такі заходи дозволяють локалізувати осередок займання і не допустити його поширення на критичне обладнання.

Окреме значення має впровадження технічних механізмів захисту, які автоматично активуються при фіксації загоряння.

• Негайне вимкнення (англ. Emergency Power Off, EPO) – система аварійного вимкнення живлення, що блокує подачу струму до серверних шаф для запобігання короткому замиканню.

• Автоматичне відключення вентиляції та кондиціонування, щоб припинити подачу кисню до зони займання та зменшити поширення диму через повітряні канали [25].

Залежно від конкретної ситуації в ЦОД, реагування може бути різним. Наприклад, якщо система виявляє лише тління ізоляції або слабке задимлення, активується лише сигнал тривоги першого рівня, відбувається локальна перевірка без повної евакуації персоналу. Якщо ж фіксується коротке замикання або перегрівання обладнання з видимим димом, може запускатися часткове гасіння газовою системою або вручну за допомогою CO_2-вогнегасника.

У випадку масового задимлення, яке поширюється вентиляцією, активується повна евакуація персоналу, система автоматично вимикає подачу повітря і запускає газове гасіння. А якщо вже виявлене відкрите полум’я або сильна пожежа, то відбувається негайне відключення живлення, запуск усіх систем гасіння та виклик пожежної служби [26].

Варто також враховувати, що не кожне спрацювання є ознакою реальної пожежі, іноді можуть бути хибні спрацьовування через пил або вологу. У таких випадках персонал перевіряє журнал подій, проводить візуальний огляд, і після підтвердження безпеки скидає тривогу.

Отже, ефективна система протипожежного захисту в ЦОД – це не лише сукупність технічних засобів, а й правильно організована логіка реагування на кожен із типових сценаріїв. Такий комплекс підходів дозволяє не лише вчасно виявити потенційну загрозу, а й відреагувати з мінімальними ризиками для обладнання, даних і людей.

ВИСНОВКИ

У виконаній кваліфікаційній роботі детально досліджено задачу дискретного логарифму в класичних мультиплікативних групах та на еліптичних кривих, проаналізовано її математичні передумови і комплексно розглянуто основні методи атак - повний перебір, метод Шенкса, алгоритм Полларда "По", алгоритм індексного числення та атаку Поліг-Хеллмана, а також специфіку зламу ECDLP. Порівняльне оцінювання їхньої складності показало, що неправильно підібрані параметри роблять дискретний логарифм суттєво вразливим, і дозволило сформулювати практичні рекомендації: для груп Z_p^* бажано використовувати модулі не менше 3072 біт, а для еліптичних кривих – ключі від 256 біт у групах простого порядку без дрібних простих множників.

Особливий акцент зроблено на методі Поліг-Хеллмана, для якого розроблено та протестовано власну програмну реалізацію. Тестування цієї програми підтвердили, що за наявності малофакторного порядку групи час зламу скорочується на декілька порядків порівняно з перебором чи методом Шенкса. Розроблене програмне рішення дає можливість оперативно перевіряти криптостійкість реалізацій Diffie-Hellman, ElGamal і DSA, а також слугує ефективним навчально-дослідницьким інструментом, який наочно демонструє небезпеку слабких налаштувань і допомагає уникнути критичних помилок у прикладній криптографії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Christof Paar Jan. Understanding cryptography: a textbook for students and practitioners. 2nd ed. Berlin : Springer, 2010. 372 p.

2. Dummit D. S. Foote R. M. Abstract algebra. 3rd ed. Hoboken, NJ : Wiley, 2004. 932 p.

3. Gallian Joseph A. Contemporary abstract algebra. 9th ed. Boston, MA : Cengage Learning, 2017. 654 p.

4. Nils Fleischhacker Tibor Jager Dominique Schröder. On tight security proofs for schnorr. Journal of cryptology. 2019.

5. Загородна Н. В. Основи криптографії відкритих ключів. Електронне навчання ТНТУ. URL: https://dl.tntu.edu.ua/content.php?cid=415159 (дата звернення: 22.05.2025).

6. Що таке проблема дискретного логарифмування еліптичної кривої (ECDLP) і чому її важко вирішити? - Академія EITCA. EITCA Academy. URL: https://uk.eitca.org/cybersecurity/eitc-is-acc-advanced-classical-cryptography/elliptic-curve-cryptography/introduction-to-elliptic-curves/examination-review-introduction-to-elliptic-curves/what-is-the-elliptic-curve-discrete-logarithm-problem-ecdlp-and-why-is-it-difficult-to-solve/ (дата звернення: 01.06.2025).

7. A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography. Proceedings of the first IEEE international conference on sensor and ad hoc communications and networks (SECON) : conference proceedings, Santa Clara, California, 4–7 October 2004.

8. Neal Koblitz. A course in number theory and cryptography. 2nd ed. New York, USA : Springer, 2012. 235 p.

9. Contributors to Wikimedia projects. Baby-step giant-step - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Baby-step_giant-step (date of access: 14.05.2025).

10. Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone. Handbook of Applied Cryptography. Boca Raton : CRC Press, 1996. 780 p.

11. Contributors to Wikimedia projects. Elliptic-curve cryptography - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Elliptic-curve_cryptography (date of access: 20.05.2025).

12. Darrel Hankerson, Alfred J. Menezes, Scott A. Vanstone. Guide to elliptic curve cryptography. New York : Springer, 2004. 312 p.

13. NIST SP 800-57 Part 1 Rev. 5. Recommendation for key management. Replaces NIST SP 800-57 Part 1 Rev. 4 ; effective from 2020-05-01. Official edition. Gaithersburg : U.S. Department of Commerce, 2020. 85 p.

14. FIPS 186-5. Digital signature standard. Replaces FIPS 186-4 ; effective from 2023-02-03. Official edition. Gaithersburg : U.S. Department of Commerce / NIST, 2023.

15. RFC 7748. Elliptic Curves for Security. Effective from 2016-01-22. Official edition.

16. Android security vulnerability. Bitcoin.org. URL: https://bitcoin.org/en/alert/2013-08-11-android (date of access: 11.03.2025).

17. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. Advances in cryptology – CRYPTO ’96 (16th annual international cryptology conference) : Conference paper, Santa Barbara, California.

18. Учасники проектів Вікімедіа. Алгоритм Шора – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Алгоритм_Шора (дата звернення: 27.05.2025).

19. NIST Releases First 3 Finalized Post-Quantum Encryption Standards. NIST. URL: https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards (date of access: 28.05.2025).

20. Weiterleitungshinweis. Google. URL: https://www.google.com/url?sa=i&url=https://www.cryptologie.net/article/360/how-to-backdoor-diffie-hellman-quick-explanation/&psig=AOvVaw1wnxH1lefXU9oZWEhO83ba&ust=1749841918811000&source=images&cd=vfe&opi=89978449&ved=0CBMQjhxqFwoTCKC35LDL7I0DFQAAAAAdAAAAABAK (date of access: 30.03.2025).

21. Hoffstein J. Pipher J. Silverman J. H. Introduction to Mathematical Cryptography. 2nd ed. NY, USA : Springer, 2010. 524 p.

22. Yesin, V., Karpinski, M., Yesina, M., Vilihura, V., Kozak, R., Shevchuk, R. (2023). Technique for Searching Data in a Cryptographically Protected SQL Database. Applied Sciences, 13(20), art. no. 11525, 1-21. doi: 10.3390/app132011525.

23. Skorenkyy, Y., Kozak, R., Zagorodna, N., Kramar, O., & Baran, I. (2021, March). Use of augmented reality-enabled prototyping of cyber-physical systems for improving cyber-security education. In Journal of Physics: Conference Series (Vol. 1840, No. 1, p. 012026). IOP Publishing.

24. Kuznetsov, O., Poluyanenko, N., Frontoni, E., Kandiy, S., Karpinski, M., & Shevchuk, R. (2024). Enhancing Cryptographic Primitives through Dynamic Cost Function Optimization in Heuristic Search. Electronics, 13(10), 1-52. doi: 10.3390/electronics13101825.

25. Деркач М. В., Мишко О. Є. Використання алгоритму шифрування AES-256-CBC для зберігання даних автентифікації автономного помічника. Наукові вісті Далівського університету. 2023. №24.

26. Гульчак Ю. П. Северин Л. І. Основи інженерної психології. Вінниця : ВНТУ, 2004. 105 с.

27. Брусенцов В. Г., Савченко С. О., Снігур В. І. Основи ергономіки. Харків : УкрДАЗТ, 2011. 141 с.

28. ДБН В.2.5-56:2014. Системи протипожежного захисту. Чинний від 2015-07-01. Вид. офіц. Київ, 2014.

29. Слуцька О. М., Скоробагатько Т. М., Скоробагатько Т. М. Необхідність удосконалення вимог до систем пожежної сигналізації. Пожежна безпека. 204. С. 70–78.

30. Навчально-методичний посібник до практичних заняття з дисципліни «Безпека життєдіяльності, основи охорони праці» для студентів освітнього ступеня „бакалавр" усіх спеціальностей та форм навчання / О. Я. Гурик та ін. Тернопіль : ТНТУ ім. Ів. Пулюя, 2025. 123 с.

Додаток А Програмна реалізація Pohlig-Hellman

```
from math import sqrt, ceil
def PrimeFactorization(p):
    """Find all prime factors of the number p"""
    d, primeFactors = 2, []
    while d * d <= p:
        while (p % d) == 0:
            primeFactors.append(d)
            p //= d
        d += 1
    if p > 1:
        primeFactors.append(p)
    return primeFactors

def CountOccurences(primeFactors):
    """Count occurrences of each prime factor"""
    return [[x, primeFactors.count(x)] for x in set(primeFactors)]

def ExtendedGCD(a, b):
    """Extended Euclidean algorithm for finding GCD"""
    a2, a1 = 1, 0
    b2, b1 = 0, 1
    while b:
        q, r = divmod(a, b)
        a1, a2 = a2 - q * a1, a1
        b1, b2 = b2 - q * b1, b1
        a, b = b, r
    return a, a2, b2

def ModularInverse(b, n):
    """Return x such that x ≡ a^(-1) (mod n)"""
    g, x, _ = ExtendedGCD(b, n)
    if g == 1:
        return x % n

def ChineseRemainder(pairs):
```

```
    """Chinese remainder theorem for solving a system of
congruences"""
    N, X = pairs[0][1], 0
    for ni in pairs[1:]:
        N *= ni[1]
    for (ai, ni) in pairs:
        mi = (N // ni)
        X += mi * ai * ExtendedGCD(mi, ni)[1]
    return X % N

def ShanksAlgorithm(alpha, beta, n):
    """Return x such that beta ≡ alpha^(x) (mod n)"""
    m = int(ceil(sqrt(n - 1)))
    a = pow(alpha, m, n)
    b = ExtendedGCD(alpha, n)[1]
    L1 = [(j, pow(a, j, n)) for j in range(0, m)]
    L2 = [(i, beta * (b ** i) % n) for i in range(0, m)]
    L1.sort(key=lambda tup: tup[1])
    L2.sort(key=lambda tup: tup[1])
    i, j, Found = 0, 0, False
    while (not Found) and (i < m) and (j < m):
        if L1[j][1] == L2[i][1]:
            return m * L1[j][0] + L2[i][0] % n
        elif abs(L1[j][1]) > abs(L2[i][1]):
            i += 1
        else:
            j += 1

def CongruencePair(g, h, p, q, e, e1, e2):
    """Return pair (x, q^e) which represents one congruence"""
    alphaInverse = ModularInverse(e1, p)
    x = 0  # x = x_{0} + x_{1} * q + x_{2} * q^{2} + ... + x_{e -
1} * q^{e - 1}
    for i in range(1, e + 1):
        a = pow(e1, q ** (e - 1), p)
        b = pow(e2 * (alphaInverse ** x), q ** (e - i), p)
        x += ShanksAlgorithm(a, b, p) * (q ** (i - 1))
```

```
    return (x, q ** e)

def PrintFormated(arg1, arg2, arg3, arg4, arg5):
    print(" {:3s} | {:3s} | {:13s} | {:13s} |
{:45s}".format(str(arg1), str(arg2), str(arg3), str(arg4),
str(arg5)))
    print("-" * 90)

def PohlingHellman(g, h, p):
    """Main function of Pohling-Hellman's algorithm"""

    CountOccurencesList = CountOccurences(PrimeFactorization(p -
1))
    CongruenceList = []

    print("\n")
    print("-" * 90)
    print(f" Solving {g} ≡ {h}^x (mod {p})")
    print("-" * 90)
    PrintFormated("q", "e", "g^((p-1)/q^e)", "h^((p-1)/q^e)",
"Solve (g^((p-1)/q^e))^x = h^((p-1)/q^e) for x")

    for i in range(len(CountOccurencesList)):
        e1 = (h ** ((p - 1) // (CountOccurencesList[i][0] **
CountOccurencesList[i][1]))) % p  # e1 = g^((p-1)/q^e)
        e2 = (g ** ((p - 1) // (CountOccurencesList[i][0] **
CountOccurencesList[i][1]))) % p  # e2 = h^((p-1)/q^e)
        # Add new congruence
        CongruenceList.append(CongruencePair(g, h, p,
CountOccurencesList[i][0], CountOccurencesList[i][1], e1, e2))
        e3 = CongruenceList[len(CongruenceList) - 1][0] %
CongruenceList[len(CongruenceList) - 1][1]  # e3 = (g^((p-
1)/q^e))^x
        e4 = CongruenceList[len(CongruenceList) - 1][1]  # e4 =
h^((p-1)/q^e)
        PrintFormated(CountOccurencesList[i][0],
CountOccurencesList[i][1], e1, e2, f"x ≡ {e3} (mod {e4})")
```

```
    # Solve the system of congruences
    print(f" Solution x = {ChineseRemainder(CongruenceList)}")
    print("-" * 90)
    print("\n")

if __name__ == '__main__':
    print("\nPress CTRL + C to exit\n")
    print("=" * 90)
    print("Pohling-Hellman's algorithm for discrete logarithm")
    print("Formula : h ≡ g^x (mod p)")
    print("=" * 90)
    print("\n")  # TEST EXAMPLES (h, g, p) SOLUTIONS
    # Example: PohlingHellman(18, 2, 29) returns 11
    while True:
        err = False
        h = int(input("h: "))  # Use input() for Python 3
        g = int(input("g: "))  # Use input() for Python 3
        p = int(input("p: "))  # Use input() for Python 3

        if p < 2:
            print("Group order must be greater than one.\n")
            err = True

        if not err:
            try:
                PohlingHellman(h, g, p)
            except TypeError:
                print(f" The congruence {h} ≡ {g}^x (mod {p}) has
no solution ")
                print("-" * 90)
                print("\n")
```